



# Simulation Modeling with Simio

## a Workbook

5th Edition

Jeffrey Allen Joines  
Michael E. Kuhl



# SIMULATION MODELING WITH SIMIO: A WORKBOOK V5

Jeffrey Allen Joines

North Carolina State University

Raleigh, North Carolina

Michael E. Kuhl

Rochester Institute of Technology

Rochester, New York

July 2024

5<sup>th</sup> Edition

Copyright 2024 by Jeffrey A. Joines and Michael E. Kuhl. All rights reserved.

Electronic Book ISBN-13: 978-1-938207-07-5, ISBN-10: 19382070750

Published by:  
SIMIO LLC  
504 Beaver St, Sewickley, PA 15143, USA  
<http://www.simio.com>





# Table of Contents

---

<b>About the Authors .....</b>	<b>v</b>
PART 1: WHAT MAKES THIS BOOK DIFFERENT? .....	VI
PART 2: WHO BENEFITS FROM ACTIVE LEARNING? .....	VI
PART 3: WHY EMPHASIZE “SIMULATION MODELING?” .....	VII
PART 4: ORGANIZATION OF THIS BOOK .....	VII
PART 5: CHANGES IN THE BOOK .....	VIII
PART 6: STYLES USED IN THIS BOOK .....	VIII
PART 7: ACKNOWLEDGEMENTS .....	VIII
 <b>Chapter 1 Introduction to Simulation: The Ice Cream Store.....</b>	 <b>1</b>
PART 1.1: WHAT IS SIMULATION?.....	1
PART 1.2: SIMULATION FUNDAMENTALS: THE ICE CREAM STORE .....	3
PART 1.3: MANUAL SIMULATION.....	10
PART 1.4: INPUT MODELING AND SIMULATION OUTPUT ANALYSIS.....	14
PART 1.5: ELEMENTS OF THE SIMULATION STUDY .....	20
PART 1.6: COMMENTARY.....	20
 <b>Chapter 2 Introduction to SIMIO: The Ice Cream Store.....</b>	 <b>22</b>
PART 2.1: GETTING STARTED.....	22
PART 2.2: THE ICE CREAM STORE.....	24
PART 2.3: ENHANCING THE ANIMATION.....	28
PART 2.4: LOOKING AT THE RESULTS.....	30
PART 2.5: COMMENTARY.....	32
 <b>Chapter 3 Modeling Distance and Examining Inputs/Outputs .....</b>	 <b>34</b>
PART 3.1: BUILDING THE MODEL.....	34
PART 3.2: USING THE 3D WAREHOUSE .....	35
PART 3.3: EXAMINING MODEL INPUT PARAMETERS.....	37
PART 3.4: EXAMINING OUTPUT.....	38
PART 3.5: USING EXPERIMENTS .....	40
PART 3.6: INPUT SENSITIVITY .....	43
PART 3.7: COMMENTARY.....	44
 <b>Chapter 4 More Detailed Modeling: Airport Revisited.....</b>	 <b>45</b>
PART 4.1: CHOICE OF PATHS.....	45
PART 4.2: CHANGING ARRIVAL RATE .....	47
PART 4.3: STATE VARIABLES, PROPERTIES, AND DATA TABLES.....	50
PART 4.4: MORE ON BRANCHING .....	54
PART 4.5: WORK SCHEDULES .....	56
PART 4.6: COMMENTARY.....	59
 <b>Chapter 5 Data-Based Modeling: Manufacturing Cell.....</b>	 <b>60</b>
PART 5.1: CONSTRUCTING THE MODEL.....	61
PART 5.2: SETTING CAPACITIES .....	63
PART 5.3: INCORPORATING SEQUENCES.....	64
PART 5.4: EMBELLISHMENT: NEW ARRIVAL PATTERN AND PROCESSING TIMES.....	67
PART 5.5: USING RELATIONAL TABLES.....	71
PART 5.6: CREATING STATISTICS .....	74
PART 5.7: OBTAINING STATISTICS FOR ALL PART TYPES .....	77
PART 5.8: AUTOMATING THE CREATION OF ELEMENTS FOR STATISTICS COLLECTION.....	79
PART 5.9: COMMENTARY.....	84
 <b>Chapter 6 Assembly and Packaging: Memory Chip Boards.....</b>	 <b>85</b>
PART 6.1: MEMORY BOARD ASSEMBLY AND PACKING .....	85

PART 6.2: MAKING THE ANIMATION REVEAL MORE INFORMATION .....	91
PART 6.3: EMBELLISHMENT: OTHER RESOURCE NEEDS.....	93
PART 6.4: CHANGING PROCESSING TIME AS A FUNCTION OF THE SIZE OF THE QUEUE .....	96
PART 6.5: CREATING STATISTICS .....	98
PART 6.6: COMMENTARY .....	100
<b>Chapter 7 Using SIMIO Processes .....</b>	<b>101</b>
PART 7.1: THE ADD-ON PROCESS .....	101
PART 7.2: THE ADD-ON PROCESS TRIGGERS: ILLUSTRATING “ASSIGN” STEP .....	102
PART 7.3: CREATING AN INDEPENDENT “REUSABLE” PROCESS .....	104
PART 7.4: COLLECTING TALLY STATISTICS.....	105
PART 7.5: MANIPULATING RESOURCES .....	109
PART 7.6: TOKENIZED PROCESSES.....	110
PART 7.7: COMMENTARY .....	113
<b>Chapter 8 Working with Flow and Capacity: The DMV.....</b>	<b>114</b>
PART 8.1: THE DMV OFFICE.....	115
PART 8.2: USING RESOURCES WITH SERVERS.....	120
PART 8.3: HANDLING FAILURES .....	121
PART 8.4: SERVER CONFIGURATION ALTERNATIVES .....	124
PART 8.5: RESTRICTING ENTITY TO FLOW TO PARALLEL SERVERS.....	126
PART 8.6: THE WAITING ROOM SIZE .....	132
PART 8.7: USING APPOINTMENT SCHEDULES .....	134
PART 8.8: CONTROLLING THE SIMULATION REPPLICATION LENGTH .....	140
PART 8.9: COMMENTARY .....	144
<b>Chapter 9 The Workstation Concept: A Kitting Process.....</b>	<b>145</b>
PART 9.1: THE KITTING PROCESS.....	145
PART 9.2: MODELING BATCHING, SETUP, AND TEARDOWN TIMES.....	149
PART 9.3: SEQUENCE-DEPENDENT SETUP TIMES .....	152
PART 9.4: SEQUENCE-DEPENDENT SET-UP TIMES THAT ARE RANDOM .....	155
PART 9.5: USING MATERIALS IN THE KITTING OPERATION .....	158
PART 9.6: RAW MATERIAL ARRIVALS DURING THE SIMULATION .....	160
PART 9.7: IMPLEMENTING A JUST-IN-TIME APPROACH.....	162
PART 9.8: COMMENTARY .....	167
<b>Chapter 10 Inventories, Supply Chains, and Optimization .....</b>	<b>168</b>
PART 10.1: BUILDING A SIMPLE SUPPLY CHAIN .....	168
PART 10.2: PROCESSING ORDERS IN THE SUPPLY CHAIN SYSTEM.....	172
PART 10.3: CREATING THE REPLENISHMENT PART OF THE SUPPLY CHAIN SYSTEM .....	174
PART 10.4: USING AN EXPERIMENT TO DETERMINE THE BEST VALUES .....	177
PART 10.5: USING SMORE PLOTS TO DETERMINE THE BEST VALUES.....	178
PART 10.6: USING RANKING AND SELECTION TO DETERMINE THE BEST SCENARIO.....	180
PART 10.7: USING OPTQUEST™ TO OPTIMIZE THE PARAMETERS .....	182
PART 10.8: MULTI-OBJECTIVE AND ADDITIONAL CONSTRAINTS USING OPTQUEST™ .....	185
PART 10.9: COMMENTARY .....	189
<b>Chapter 11 Simulation Output Analysis.....</b>	<b>190</b>
PART 11.1: WHAT CAN GO WRONG? .....	190
PART 11.2: TYPES OF SIMULATION ANALYSES .....	191
PART 11.3: OUTPUT ANALYSIS.....	191
PART 11.4: AUTOMATIC BATCHING OF OUTPUT .....	196
PART 11.5: ALGORITHMS USED IN SIMIO BATCH MEANS METHOD.....	197
PART 11.6: INPUT ANALYSIS.....	198
PART 11.7: COMMENTARY .....	203
<b>Chapter 12 Materials Handling.....</b>	<b>204</b>

PART 12.1: VEHICLES: CART TRANSFER IN MANUFACTURING CELL.....	204
PART 12.2: CART TRANSFER AMONG STATIONS.....	209
PART 12.3: OTHER VEHICLE TRAVEL BEHAVIORS (FIXED ROUTE AND FREE SPACE TRAVEL) .....	216
PART 12.4: CONVEYORS: A TRANSFER LINE .....	219
PART 12.5: MACHINE FAILURES IN THE CELL.....	223
PART 12.6: SORTING CONVEYORS.....	224
PART 12.7: COMMENTARY.....	226
<b>Chapter 13 Management of Resources: Veterinary Clinic.....</b>	<b>227</b>
PART 13.1: UTILIZING THE FIXED RESOURCE OBJECT .....	227
PART 13.2: DIFFERENT RESOURCE NEEDS BASED ON DIFFERENT PATIENT TYPES.....	231
PART 13.3: RESOURCE DECISION MAKING .....	237
PART 13.4: ADDING AN ADDITIONAL PROCESS .....	238
PART 13.5: CHANGING PROCESSING BASED ON ANIMAL TYPE AND VET SERVICING .....	241
PART 13.6: CHANGING THE RESOURCE ALLOCATION SELECTION.....	247
PART 13.7: COMMENTARY.....	249
<b>Chapter 14 A Mobile Resource: The Worker .....</b>	<b>250</b>
PART 14.1: ROUTING PATIENTS .....	250
PART 14.2: USING A WORKER AS A RESOURCE .....	253
PART 14.3: USING A WORKER AS A MOVEABLE RESOURCE .....	257
PART 14.4: RETURNING TO THE OFFICE BETWEEN PATIENTS.....	260
PART 14.5: ZERO-TIME EVENTS.....	264
PART 14.6: HANDLING MULTIPLE VETS.....	266
PART 14.7: COMMENTARY.....	268
<b>Chapter 15 Adding Detail to Service: A Bank Example.....</b>	<b>270</b>
PART 15.1: USING A WORKER AS A RESOURCE AND A VEHICLE .....	270
PART 15.2: HAVING THE BANKER ESCORT THE CUSTOMER TO THE SAFETY DEPOSIT BOX ROOM .....	274
PART 15.3: USING THE TRANSPORT FUNCTION OF THE WORKER.....	279
PART 15.4: RESOURCE RESERVATIONS .....	282
PART 15.5: ANIMATED ENTITIES.....	284
PART 15.6: DETAILED SERVICE: TASKS AND TASK SEQUENCES.....	286
PART 15.7: USING TASK SEQUENCES .....	288
PART 15.8: SOME OBSERVATIONS CONCERNING TASKS.....	291
PART 15.9: COMMENTARY.....	297
<b>Chapter 16 Modeling of Call Centers.....</b>	<b>298</b>
PART 16.1: BUILDING THE SIMPLE MODEL.....	299
PART 16.2: BALKING .....	302
PART 16.3: MODELING RENEGING OF CUSTOMER CALLS .....	304
PART 16.4: OPTIMIZING THE NUMBER OF FIRST-LINE TECHNICIANS .....	308
PART 16.5: USING THE FINANCIAL COSTS AS THE OPTIMIZING OBJECTIVE.....	312
PART 16.6: COMMENTARY.....	316
<b>Chapter 17 Sub-Modeling: Cellular Manufacturing .....</b>	<b>318</b>
PART 17.1: MODEL OF ONE WORK CELL.....	318
PART 17.2: CREATING THE SUB-MODEL.....	320
PART 17.3: CREATING A MODEL USING THE WORKCELL SUB-MODEL .....	322
PART 17.4: ADDING TO THE WORKCELL OBJECT .....	324
PART 17.5: EXPOSING RESOURCE AND CAPACITY PROPERTIES .....	326
PART 17.6: PASSING INFORMATION BETWEEN THE MODEL AND ITS SUB-MODELS .....	328
PART 17.7: COMMENTARY.....	330
<b>Chapter 18 The Anatomy of Objects: Server.....</b>	<b>331</b>
PART 18.1: A SIMPLE RESOURCE MODEL: WAREHOUSE PICKUP.....	331
PART 18.2: TAKING AN OBJECT APART TO FIGURE OUT HOW IT WORKS.....	334

PART 18.3: SIMIO OBJECTS AND CLASS HIERARCHY .....	338
<b>Chapter 19 Building New Objects via Sub-Classing: A Delay Object.....</b>	<b>341</b>
PART 19.1: SUB-CLASSING THE TRANSFERNODE TO CREATE A DELAYNODE.....	341
PART 19.2: MODIFYING PROCESSES AND ADDING PROPERTIES FOR THE NEW NODE .....	343
PART 19.3: CREATING A MODEL TO TEST THE NEW DELAYTRANSFERNODE.....	345
PART 19.4: COMMENTARY .....	350
<b>Chapter 20 Creating New Objects.....</b>	<b>351</b>
PART 20.1: CREATING A SIMPLE DELAY OBJECT .....	351
PART 20.2: ADDING COLOR AND PROCESSING CONTENTS .....	355
PART 20.3: EMBELLISHING WITH USER-DEFINED ADD-ON PROCESS TRIGGERS .....	357
PART 20.4: EMBELLISHING WITH STATE ASSIGNMENTS.....	360
PART 20.5: ADDING SECONDARY RESOURCES.....	364
PART 20.6: USING STORAGES TO DISTINGUISH WAITING VERSUS DELAYING .....	366
PART 20.7: CONSIDERING CAPACITY LIKE A SERVER .....	369
PART 20.8: COMMENTARY .....	371
<b>Chapter 21 Continuous Variables, Reneging, Interrupt, Debugging: A Gas Station .....</b>	<b>372</b>
PART 21.1: SIMPLE TANK MANUAL PROCESS.....	372
PART 21.2: SIMPLE TANK REVISITED USING THE FLOW LIBRARY .....	376
PART 21.3: THE GAS STATION.....	379
PART 21.4: RENEGING THE CARS WHEN THE PUMP GOES OFF.....	385
PART 21.5: INTERRUPTING THE CARS WHEN THE PUMP GOES OFF .....	387
PART 21.6: USING ENTITIES THAT CARRY A TANK .....	388
PART 21.7: COMMENTARY .....	392
<b>Chapter 22 More Subclassing: Advanced Modeling of Supply Chain Systems .....</b>	<b>393</b>
PART 22.1: DEVELOPING A SPECIALIZED SUPPLY CHAIN WORKSTATION OBJECT.....	393
PART 22.2: ADDING THE ORDERING STATION AND CHARACTERISTICS TO HANDLE ORDERS.....	395
PART 22.3: ADDING THE BEHAVIOR LOGIC FOR THE ORDERING SYSTEM.....	399
PART 22.4: ADDING THE BEHAVIOR LOGIC FOR THE INVENTORY REPLENISHMENT SYSTEM.....	402
PART 22.5: USING THE NEW TO MODEL THE COMPLEX SUPPLY SYSTEM.....	406
PART 22.6: ADDING A SECONDARY SUPPLIER FOR OVERFLOW ORDERS .....	409
PART 22.7: COMMENTARY .....	411
<b>Chapter 23 More Subclassing: Process Planning/Project Management.....</b>	<b>413</b>
PART 23.1: PROCESS PLANNING .....	413
PART 23.2: CREATING A SPECIALIZED TIMEPATH TO HANDLE ACTIVITIES.....	414
PART 23.3: CREATING A JUNCTION OBJECT TO HANDLE PRECEDENT CONSTRAINTS.....	416
PART 23.4: CREATING SMALL NETWORK TO TEST THE NEW OBJECT.....	421
PART 23.5: BUILDING THE EXAMPLE NETWORK .....	423
PART 23.6: ADDING THE SLACK AND PERCENT OF TIME ON CRITICAL PATH CALCULATIONS .....	425
PART 23.7: ADDING SLACK AND PERCENT OF TIME ON CP CALCULATIONS SECOND APPROACH .....	428
PART 23.8: ERROR CHECKING TO MAKE SURE MODELER USES JUNCTION CORRECTLY.....	430
PART 23.9: COMMENTARY .....	432
<b>Appendix A Input Modeling .....</b>	<b>433</b>
PART A.1 RANDOM VARIABLES .....	433
PART A.2 COLLECTING DATA .....	433
PART A.3 INPUT MODELING: SELECTING A DISTRIBUTION TO FIT YOUR DATA .....	435
PART A.4 DISTRIBUTION SELECTION HIERARCHY .....	440
PART A.5 EMPIRICAL DISTRIBUTIONS IN SIMIO .....	440
PART A.6 SOFTWARE FOR INPUT MODELING.....	442
PART A.7 THE LOGNORMAL DISTRIBUTION .....	443
PART A.8 MODELING THE SUM OF $N$ INDEPENDENT RANDOM VARIABLES .....	444
<b>Index.....</b>	<b>447</b>

## About the Authors

---



**JEFFREY A. JOINES** is a Professor and the Department Head of the Department of Textile Engineering, Chemistry, and Science at N.C. State University. He received a B.S. in Electrical Engineering, B.S. in Industrial Engineering, an M.S. in Industrial Engineering, and a Ph.D. in Industrial Engineering from N.C. State University. He received the 1997 Pritsker Doctoral Dissertation Award from the Institute of Industrial Engineers. He is a member of IEEE, IISE, ASEE, Tau Beta Pi, Eta Kappa Nu, Alpha Pi Mu, and Phi Kappa Phi. His research interests include utilizing computer simulation and computational optimization methods in supply chain optimization. Dr. Joines teaches graduate and undergraduate classes in computer information systems, computer-based modeling in Excel and VBA, Lean Six Sigma, and computer simulation modeling. Dr. Joines has also taught industry programs in Design for Six Sigma, Simulation and Six Sigma, and Data Management to Assist in Six Sigma through the Zeis Textile Extension programs Six Sigma Black Belt and Master Black Belt program. Dr. Joines served as the Program Chair for the 2005 Winter Simulation Conference (WSC) and the Proceedings Editor for the 2000 WSC, as well as developed and maintained the WSC paper management system from 2000-2009. He served on the WSC Board of Trustees, representing the IEEE Systems, Man, and Cybernetics Society from 2010 to 2020. He has also been an author, session chair, and track chair for several Winter Simulation Conferences. He received the 2014 INFORMS Distinguished Service Award and the 2021 James R. Wilson Board of Directors Award for his service to the simulation community.

Dr. Joines is involved in utilizing technology in the classroom and how it impacts problem-solving. He was awarded the 2006 Alumni Association NC State University Outstanding Teaching Award, the 2009 Gertrude Cox Teaching Award for Innovative Excellence in Teaching and Learning with Technology for Large Transformative Projects (with Steve Roberts), and the 2012 Alumni Distinguished Undergraduate Professor award. In 2016, Joines became the first faculty member in the College of Textiles to receive the prestigious UNC System Board of Governors Awards for Excellence in Teaching.



**MICHAEL E. KUHL** is a Professor in the Department of Industrial and Systems Engineering at Rochester Institute of Technology (RIT). Professor Kuhl received his B.S. degree in Industrial Engineering from Bradley University, an M.S. degree in Industrial Engineering/Operations Research, and a Ph.D. in Industrial Engineering from North Carolina State University. His primary teaching and research interests are in simulation modeling and analysis with application to a wide range of areas, including manufacturing systems, healthcare systems, intelligent material handling systems, and cyber security. He has been a faculty member at Rochester Institute of Technology since 2001, where he served as the director of ISE Graduate Programs and twice as interim department head. Prior to joining (RIT), he was a faculty member in the Department of Internal and Manufacturing Systems Engineering at Louisiana State University.

Dr. Kuhl is a member of ASEE, IISE, and INFORMS. He is an active member of the INFORMS Simulation Society, having served in leadership roles, including President and chairing the Simulation Research Workshop (2009) in Warwick, England. In addition, he has served on the Winter Simulation Conference (WSC) Board of Directors (2016-2023). He has also served WSC as Program Chair (2013), Proceeding Editor (2005), WSC Mobile App Chair (2014-present). He has also been an author, session chair, and track chair for several Winter Simulation Conferences. In addition, he has served as a Director of the IISE Modeling and Simulation Division (2018-2020). He has been a member of the MHI College Industry Council on Material Handling Education (CICMHE) since 2020 and currently serves as Vice-President/President-Elect.

Professor Kuhl is committed to student-centered education. In particular, he has developed and adopted active learning methods aimed at fostering an entrepreneurial mindset (E.M.) in students. To this end, he is engaged with the Kern Entrepreneurial Engineering Network (KEEN) and their E.M. framework of developing aspects of curiosity, connections, and creating value. He has contributed several E.M. learning activities related to simulation concepts to KEEN.

Most books are written in an expository style in which the author(s) goes through a great deal of trouble explaining the concepts and ideas being presented in detail. This style is also prominent in simulation books, of which there are basically two categories – those that base their book on a simulation language and those that don’t. If the simulation book is based on a simulation language, then the various features and uses of the language have to be described along with the fundamental concepts in simulation. Simulation books that are not based on a language are generally better able to concentrate on the fundamentals of simulation, especially the statistical or analytical aspects. **Note the book was created using version 16.255 of SIMIO.**

## Part 1: What makes this book different?

This book is different than most. First, it’s written in what might be called a participatory style. You don’t sit and read the book without a computer loaded with SIMIO. This book expects your active participation in using SIMIO as you turn the pages. We try to carry on a conversation with you. We believe that simulation is not a spectator sport. You have to practice to gain skills with them and develop them through modeling practice. This book encourages you to practice and use your skills.

Secondly, this book is focused on simulation modeling with SIMIO, and most of the statistical analysis and analytical issues are left as topics to be explored elsewhere. Now, it’s not that we don’t think these are important because they are vital (and when we teach simulation, they are a fundamental part of the course). But we aren’t going to spend much time on these topics here since there are excellent simulation books that can fill our omission. We strongly suggest that if you are teaching/learning simulation, you also have one of the non-language books available.<sup>1</sup>

Third, this book is deliberately cheap (the E-book or the paper copy). A simulation language like SIMIO will constantly change. In fact, the SIMIO developers try to have new releases (called “sprints”) about every three months. Any book that describes SIMIO will go out of date quickly, so we want to track new features and update this book fairly often. We’ll probably change the book’s content as we find better examples and approaches.

## Part 2: Who benefits from Active Learning?

Our intent is that you become an active learner and, as our title suggests, you “work” as you read. Our classroom experience is that students learn most by “doing,” so this workbook is centered around “labs” – which our students do during class. The chapters in this book generally correspond to one lab. If you are not in a classroom but want to learn SIMIO independently, we think you will find this approach attractive for self-learning. You can work through a chapter in the evening.

There are several mechanisms for incorporating active learning in a classroom. Some teachers begin the class with some kind of orientation to the problem and the modeling features. Some teachers go directly through the modeling exercise with the students. Yet others tend to let students work through the workbook at their own pace. You can give your students certain sections to do before class as a warmup to the important modeling concepts. Critical to teaching using the active learning method is the use of “teaching moments,” as opposed to lectures. Teaching moments occur when questions arise, or observations are made, where elaboration on a topic is appropriate. Some people think of this as a “pull” educational process. The approach is scary for many teachers because there is no formal lecture format, and the fear is that something won’t be “covered” in class.

To help ensure that everyone participates in this active learning process, we usually hand out a page or two filled out with questions derived from the chapter exercise at the beginning of class. We sprinkled some of these questions throughout the chapters of this book. They have short answers and require the student to pay

---

<sup>1</sup> For example, *Discrete-Event System Simulation* (5th Edition), Jerry Banks, John Carson, Barry Nelson, David Nicol, Prentice-Hall, 2010 (622 pages)

attention to what is happening. Accordingly, our classes are well attended even though we don't officially take attendance, but we do give credit to students who turn in their in-class assignments each day. Courses like this develop a reputation as a class you need to attend. *If you would like access to our materials, questions, and models, please email us, and we will grant you access to them for your teaching needs.*

### **Part 3: Why emphasize “Simulation Modeling?”**

In our view, simulation modeling is a form of “systems engineering.” Our intent is to engineer or re-engineer a system, but because that system is complex, difficult (or impossible) to experiment on, or doesn’t exist, we resort to building a model of that system on the computer and experimenting with that model (similar to how airplane designers use a wind tunnel to experiment with airplane designs). The keystone activity is “building the model,” which is what we call simulation modeling. Simulation modeling is not an exact science but draws upon the problem-solving approach. Building a simulation model of a system requires (1) a robust set of modeling concepts (the simulation language) and (2) a computer implementation. So, to become proficient at simulation modeling, you need to acquire knowledge of the modeling concepts and experience with their use.

SIMIO provides a wealth of simulation modeling concepts and features, and its implementation appeals to our need for visual and numeric results. However, anyone who has experience with simulation modeling knows that simulation languages have limits, and sometimes, we can’t build the model we want because the language limits us. Instead of the entity, attribute, and resource approach, SIMIO is based on the more general object-oriented paradigm, in which the objects execute processes. In addition to the standard objects and processes, the user can add, subtract, and change objects and processes to meet particular needs.

So, learning to use SIMIO requires different thinking. You will need to set aside the perspectives you have learned from another simulation language and begin to adopt another way to view the model-building process. If you work carefully through this book, we think you will learn enough about the SIMIO modeling approach to get to the next level, where it becomes **your** approach to simulation modeling.

### **Part 4: Organization of this book**

This book is conceptually organized, so you can build models quickly. In the first five chapters, we concentrate on the use of the Standard Library Objects in SIMIO. You can do a lot of simulation modeling without resorting to more complex concepts. Then, the next seven chapters show you how to extend the standard objects using processes. Learning how to extend the objects gives you more modeling flexibility without having to invent your own library of objects. In the later chapters, we discuss creating new objects and modifying existing objects within SIMIO and show you the power of its object-oriented capabilities.

The book is designed to be read from chapter to chapter, although it may be possible to pick out certain concepts and topics. In the beginning chapters, we construct models rather directly without much explanation as to why certain features are chosen. In the later chapters, we provide more explanations of why the modeling features were chosen and what else might be done. Some later chapters return to previously introduced topics, but we try to present them in more detail. Some redundancy is helpful in learning. By the time you have finished this book, you should be well-prepared to build models in SIMIO and understand the virtues of different modeling approaches.

At the end of most chapters, we offer commentary on the topics presented. We will emphasize the strengths and weaknesses of the modeling approach and the language (we have no financial stake in SIMIO). The designers of SIMIO were also the designers of Arena, and there are Arena fingerprints on SIMIO. However, be prepared to go beyond what you have learned in Arena or any other simulation language.

When comparing Arena to SIMIO, you will not find an Arena-like Input Analyzer or an Output Analyzer in SIMIO. SIMIO will probably never have an input analysis capability, as many third-party ones are available.<sup>2</sup> When using an input modeler, be sure the parameters being fitted from the input modeler can be converted to the parameters used in SIMIO. The Output Analyzer function and the Process Analyzer in Arena are extended in SIMIO, with ways to write information externally to files for other analysis. SIMIO's output display is improving on an almost daily basis. SIMIO has incorporated recent research into subset selection and the ranking/selection of scenarios. Optimization features have also been added. Further, SIMIO has 3D graphics built to scale and possesses an object orientation that allows new objects to be added and processes to be easily changed. SIMIO has a far more modern “look and feel” and, in our opinion, is generally superior to most other simulation language choices.

## Part 5: Changes in the Book

### Fifth Edition:

- The biggest change is Michael E. Kuhl has helped with the revision of the new book. Stephen Roberts has finally retired and wants to spend more time with his family, playing golf and doing woodworking.
- In any type of software book, it is not until the book is actually used that mistakes crop up. This edition corrects several mistakes and typos that occurred in the fourth edition.
- Also, more clarifications and figures were added to make things clearer. There are discussions and figures on how `TOKENS` are used to execute process steps.
- We find related tables to be quite a useful modeling technique, and they are used in several chapters now.
- SIMIO is a language that is updated frequently through the use of sprints. This edition of the book utilized version 16.255 of the software, and all the figures and concepts have been updated. In the last few years, SIMIO has introduced several new modeling concepts and made changes to other concepts. For example, the Workstation has been deprecated and is no longer being developed owing to the modifications to the Task Sequences. The chapters that utilized `WORKSTATION` now utilize the task sequences or the new function `Math.SumofSamples` to do many of the `WORKSTATION` features.

## Part 6: Styles Used in this Book

Certain styles have been used in this book to illustrate objects, names, and parameters and make distinguishing these types of parameters easier. Standard SIMIO objects will be set in small caps using a Courier New font (e.g., `SERVER`), while objects the modeler creates will also be bolded (e.g., `DELAYOBJECT`). Properties associated with these objects will be italicized (e.g., *Processing Time*). Process names will be italicized and placed in quotes, such as “OnEnteredProcessing,” while add-on process triggers (e.g., `Exited`) will only be italicized since they are properties. Process steps like `Assign` will be italicized in Courier New font. SIMIO uses lots of expressions. These are set in Courier New font (e.g., `SrvOffice.Contents >0`). Names of all objects specified by the modeler will be bolded (e.g., Insert a `SERVER` named `SrvOffice`). Values associated with properties will be set in a fashion similar to expressions or in quotes for strings (e.g., “True”).

## Part 7: Acknowledgements

We wish to thank our students, who have added much to our understanding and who let us often display our ignorance and inadequate preparation. We want to thank Steve Roberts for his work on the first four editions and insight that led to this version. We appreciated the response of SIMIO developers to our endless stream of questions and doubts, especially Dave Sturrock and Dennis Pegden.

---

<sup>2</sup> For instance, see EasyFit at <http://www.mathwave.com/>

Please let us know how we can improve this workbook and how it can better meet your needs.

Jeff Joines ([JeffJoines@ncsu.edu](mailto:JeffJoines@ncsu.edu))  
North Carolina State University  
Raleigh, North Carolina

Michael Kuhl ([mekeie@rit.edu](mailto:mekeie@rit.edu))  
Rochester Institute of Technology  
Rochester, New York



# Chapter 1

## Introduction to Simulation: The Ice Cream Store

---

This chapter will give the novice in simulation an introduction to the terminology and mechanics of simulation, as well as statistics and performing a simulation model. If you are already familiar with these concepts, feel free to start with Chapter 2, which starts the simulation modeling in SIMIO™.

### Part 1.1: What is Simulation?

The word “simulation” has a variety of meanings and uses. Probably its most popular usage is in games. A video game “simulates” a particular environment, maybe a Formula One race, a battle, or a space encounter. The game allows the user to experience something similar to driving in a race, maybe with other people participating. In the military, commanders create a battlefield simulation where soldiers act according to their training, perhaps by defending a base or assaulting an enemy position. In aeronautical engineering, an engineer may take a model airplane to a wind tunnel and test its aerodynamics. Most simulations have elements of reality with the intention that the participant will learn something about the environment through the simulation or perhaps only learn to play the game better.

We employ simulation to study and improve “systems” of people, equipment, materials, and procedures. We use simulation to mimic or imitate the behavior of systems like factories, warehouses, hospitals, banks, supermarkets, theme parks – just about anywhere a service is provided or an item is being produced. Our simulations are different from gaming and training simulations in that we want the simulation to model the real system so we can investigate various changes before making recommendations. In that sense, simulation is a *performance improvement tool*. The simulation acts as an experimental laboratory, except that our laboratory is not physical but instead a computer model. We can then perform experiments on our computer model.

### Modeling

Many performance improvement tools (i.e., Lean, Six Sigma, etc.) rely on models. For example, value stream maps, spaghetti diagrams, process flow charts, waste walks, etc., are helpful conceptual/descriptive models for performance improvement based on direct observations of the system. These models provide a wide range of vehicles for describing and analyzing various systems. More formal models employ mathematical and statistical methods. For example, linear regression is a popular modeling technique in statistics. Queuing models offer a means for describing an important group of stable stochastic processes. Linear programming is a formal optimization method of finding the values of variables that minimize or maximize a linear objective function subject to linear constraints. Nevertheless, all these methods require various assumptions about the system being modeled. For example, the variables are constants (i.e., real or integer-valued) and often related linearly. If the variables have statistical variation, they are assumed to be normally or exponentially (i.e., Markovian) distributed.

Simulation is a *model-based* improvement tool. However, a few assumptions need to be made to build the model. The model can be non-linear, described by arbitrary random variables, have a complex relationship, and change with time (i.e., dynamic). In fact, the simulation model is limited only by your imagination and the nature of the system being considered. You determine the nature of the model based on what you think is important about the system being studied.

### Computer Simulation Modeling

Instead of a formal model, our simulations are *computational*. The model is essentially a logical description of how the components of a system interact. This description is translated into a computational structure within the

computer using a simulation language. The computer simulation of the system is executed over and over to generate statistical information about the system's behavior. We use statistics to describe the system's performance measures. We modify the computer simulation models to study alternative systems (i.e., experiments) based on what we learn about the system. By comparing these alternative systems statistically, we can offer performance improvement recommendations.

Of course, we could experiment directly with the system. If we thought an additional person on the assembly line would improve its production, we could try that. If we thought a new configuration of the hospital emergency room would provide more efficient care, we could create that configuration. If we thought a new inventory policy would reduce inventory, we could implement it. But now, the value of a computer simulation model becomes apparent. A change in a computer simulation model is cheaper and less risky than changing the real system. Using a computer simulation to determine if the changes are beneficial is also faster. Also, it might be safer to try a change in a simulation model than to try it in real life. In general, it is much easier to try changes in a computer model than in a real operating system, especially since changes disrupt people and facilities.

A computer simulation model allows us to develop confidence in making performance improvement recommendations. We can try out various alternatives before disrupting an existing system. You can now see why many companies and managers require that a simulation study be done before making substantial changes to any working system, especially when there are potential negative consequences of performance changes and costs that don't provide improvement.

## Verification and Validation

Since simulation models often have serious consequences and are developed with a minimum set of assumptions, their validity must be carefully considered before we believe in the recommended benefits. In simulation, we often use the words "verification" and "validation," and we have specific definitions for each.

The word "verification" refers to the model and its behavior. We most often develop our simulation models with a simulation language. This language translates our modeling "intent" into a computation structure that produces statistics. Most simulation languages can be used to describe a complex operating system, and the language provides a framework of components for viewing that system. As our models become more complicated, we employ more complex simulation language constructs to model the behavior. This relationship is critically dependent on our understanding of the simulation language intricacies. We may or may not fully understand the computational code, and thus, we confront the key question in verification: *Does the simulation model behave the way we expect?* Could we have made a mistake in employing the simulation language, or does the simulation language create an unexpected behavior in the computational code? Therefore, even after we have created our simulation model, it needs to be tested to see if it behaves as expected. If we make the processing time longer, does it result in longer times in the system? Does the waiting time decrease if we reduce the arrival rate to a queue? The answers to any of these questions are specific to the system and your model of it. But, above all else, we need to be sure that our model is behaving the way we expect without error.

The word "validation" refers to the relationship between the model and the real system (i.e., *Does the simulation produce performance measures consistent with the real system?*) Now, we are assuming our model has been verified, but can we validly infer from the real system? This question strikes at the heart of our modeling effort because we cannot legitimately say much about performance improvement without a valid model. Many people new to simulation may want the model to be a substitute for the "real system," which is a limitless task. After all, the only "model" that is perfectly representative of the real system is the system itself!

We must always remember in simulation that our model is only an *approximation* of the real system. So, the most relevant way to validate our model is to concern ourselves with its approximation. How do we decide what to approximate? It depends on our performance measures. If our performance measure is time in the

system, then we concern ourselves with those factors that impact time in the system. If our performance measure is production, then we focus on those factors that impact production. Usually, we are interested in several performance measures, but those measures will be the focus of our concern and will limit our modeling activity. Otherwise, without a clear set of performance objectives, we are left with a search for reality in our model, which is a never-ending task.

Generally, people who are only familiar with simulation think simulation models can mimic anything and often drive the development of a needlessly complex model. One of the difficult responsibilities of anyone engaged in simulation modeling is to educate the stakeholders on the benefits and limitations of a simulation model as well as the simulation modeling activity.

### **Computer Simulation Languages**

The process of creating a computer simulation model varies from programming your own model in a programming language such as C++, C#, Python, or Java to using a spreadsheet like Excel. There are a variety of simulation languages including Simula™, GPSSTM, SIMSCRIPT™, Arena/SIMANTM, SLX™, ProModel™, Flexsim™, AutoMod™, ExtendSim™, Witness™, AnyLogic™, among many others. Each simulation language differs in how it requires users to construct a simulation model. An important distinction is the degree to which computer programming is required. Also, many of the simulation languages have evolved from a particular industry or set of applications and are especially useful in that context.

We have chosen to use the simulation language SIMIO™, a relatively new language developed over the past several years. The language developers had previously developed the Arena simulation system. SIMIO benefits from more recent developments in object-oriented design and agent modeling. SIMIO is a “multi-modeling” language with agents, discrete event, and continuous language components. SIMIO was developed to provide visual appeal through its 3-D animation and graphical representations. SIMIO provides a wide range of extensions, from direct modification of executing processes to user-developed objects. SIMIO also provides interoperation with various spreadsheets and databases. Finally, SIMIO has become widely adopted in industry and academic institutions. Learning SIMIO provides one broadly-based simulation modeling tool that can help you learn to use others if needed.

### **Part 1.2: Simulation Fundamentals: The Ice Cream Store**

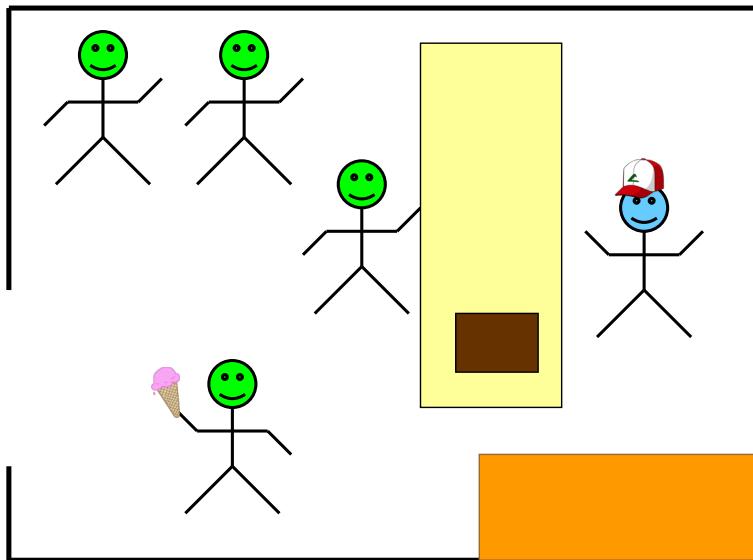
A fundamental understanding of simulation will be beneficial throughout your study of simulation. It's easy to get caught up in the creation of a computer model using a simulation language and miss important basic principles of simulation modeling. Too many people associate simulation with a simulation language; for them, simulation is simply learning the language. Learning a simulation language is necessary for using simulation, but it does not substitute for understanding at a fundamental level. If you understand the fundamentals of simulation, you establish a basis for understanding any simulation language and model. In fact, this understanding will be a key to learning almost everything else in this text.

#### **The Ice Cream Store**

It is helpful if the discussion of simulation is done in the context of a problem – albeit a simple problem. Using this simple problem, we can describe simulation elements of modeling, execution, and analysis. Our problem is the common ice cream store since everyone loves ice cream and probably has visited an ice cream store. As seen in Figure 1.1, customers arrive at the ice cream store to obtain an ice cream cone. It's a simple store where there is only one attendant, and people will wait in a single line to order and receive their ice cream cone. The attendant waits on each customer, one at a time, in the order they arrive.

*Question 1:* What might be your operational concerns if you owned or managed the ice cream store?

Likely, one of your most prominent concerns would be this store's operation and how you can improve it. For example, could you buy a new cone-making machine to make cones faster, or should you hire someone to help service customers? Should you resize the waiting space? These are performance improvement concerns.



**Figure 1.1: The Ice Cream Store**

*Question 2:* If you made one of these changes to the ice cream store, how would you decide if it improved the store's operation?

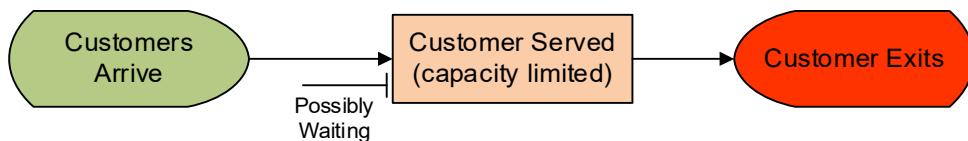
---

In other words, what are the performance measures? Here are some possible measures: the number of customers served per day, the time customers spend in the store, the waiting time of customers, the number of customers waiting, and the utilization of servers. We will discover these are common performance measures and will often be your first performance measurement choices.

*Question 3:* How can you expect to improve performance if you don't know what is happening inside the ice cream store?

---

You need to employ all the descriptive tools you know to understand what is going on inside the system. At this stage, one might create a value stream map, a flow chart, a relationship chart, a spaghetti diagram, etc. These techniques will greatly improve your understanding of what happens to customers and the attendants during the sale of an ice cream cone. Perhaps you develop the conceptual model (i.e., flow chart) presented in Figure 1.2.



**Figure 1.2: Flow Chart of the Ice Cream Store**

The conceptual model shows how the customers are served. Notice that we have added the possibility of waiting due to the fact that the availability of a single attendant limits service.

## Gathering Data about the Ice Cream Store

Descriptive information helps us understand the service process in the ice cream store; however, it doesn't give us the performance measures. For that, we need to document what is happening (i.e., we need to do a “*present systems analysis*”). Suppose we decide to do a “time study” of the store operations, as shown in Table 1.1.

**Table 1.1: Direct Observation Time Study (Event View) of Ice Cream Store**

Time	Customer	Process
0.00		Store Opens – Server Idle
0.00	1	Arrives – Start service on Customer 1 – Server Busy
8.36	1	Departs service – Server Idle
9.01	2	Arrives – Start service on Customer 2 – Server Busy
9.98	3	Arrives – Customer 3 waits
13.50	4	Arrives – Customer 4 waits
19.36	2	Departs Service – Customer 2 leaves: 3 Start service on Customer 3 – Server Busy
23.07	5	Arrives – Customer 5 waits
27.22	3	Departs service – Customer 3 leaves: 4 Start service on Customer 4 – Server Busy
33.82	4	Departs service – Customer 4 leaves: 5 Start service on Customer 5 – Server Busy
38.18	6	Arrives – Customer 6 waits
40.00		End observations

In our time study, we simply record all the “events” and the event time that occurs in the ice cream store. An event occurs when something operationally happens in the store, like an arrival of a customer or the departure of a customer from service. Also, we recorded when the attendant (i.e., server) becomes busy or idle. Here, we only observed the first 40 minutes of the store’s operation. We recognize this isn’t enough time to gain a full understanding, but we do not intend to solve a problem at this time – only to demonstrate a method.

The time study is sufficient for us to compute some performance measures, but first, it will be helpful to re-organize our time study data. We are not going to add (or subtract) any information. We are simply going to re-organize our data from an *event view* to an *entity* (i.e., customer) *view*. The event view records events, but the entity view allows us to follow our customers. The re-organized data is shown in Table 1.2, which shows when the customer arrives at the store, enters service, and leaves the store. Notice that customers five and six service and exit the store after 40 minutes.

**Table 1.2: Re-organized Event Data in Customer View From**

Customer	Arrives to Store	Enters Service	Leaves Store
1	0.00	0.00	8.36
2	9.01	9.01	19.36
3	9.98	19.36	27.22
4	13.50	27.22	33.82
5	23.07	33.82	??
6	38.18	??	??

## Performance Measure Calculations

From the entity view of the time study data, we can easily compute several performance measures.

- **Production:** Number of people served
- The number of people served is easily calculated by looking at the number of customers that have exited the system in the simulation time. In our example, four people were served in 40 minutes, or the system had a production rate of six per hour (i.e., four customers/40 minutes \* 60 minutes/hour).
- **Flowtime, Cycle Time:** Time in System
- The time in the system is calculated by averaging the difference between when the customer entered and exited the system. Again, only four customers exited the system and contributed to the average time in the system in our simple ice cream system, as seen in Table 1.3.

**Table 1.3: Calculating Average Time in the System for the Ice Cream Store**

Customer 1	Customer 2	Customer 3	Customer 4	Average Time in System
8.36-0.0	19.36-9.01	27.22-9.98	33.82-13.5	56.27/4 = 14.07 minutes

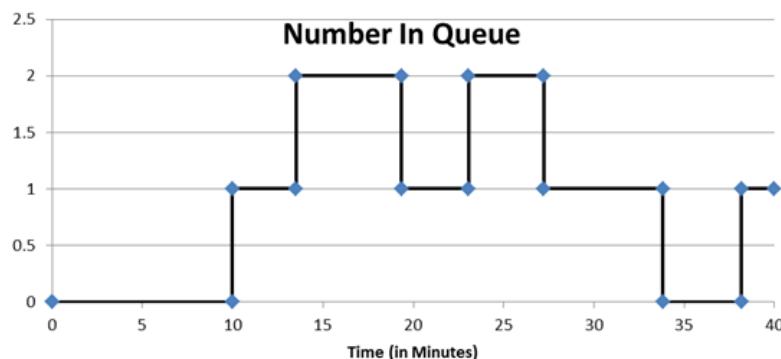
- **Non-Value Added Time:** Waiting Time in Queue  
The time in queue (i.e., the waiting or holding time) represents the time customers waited in line before being seen by the ice cream attendant. It is calculated by averaging the difference between when the customer entered the system and when they entered the service. During our observation window, five customers entered service, as seen in Table 1.4.

**Table 1.4: Calculating Average Waiting Time in the Queue**

Customer 1	Customer 2	Customer 3	Customer 4	Customer 5	Average Waiting Time
0.0-0.0	9.01-9.01	19.36-9.98	27.22-13.50	33.82-23.07	33.85/5 = 6.77 minutes

- **Number Waiting on In Queue:**

This measure determines the average number of customers one would expect to see waiting in line to receive service. Unlike the previous metrics, the number waiting in queue or number in the system measures are *time-persistent* or *time-weighted statistics*. You may have never computed a time-persistent statistic, which is a statistic for which we need to know the amount of time a value was observed. The value of the observation is weighted by the amount of time that value persists. A graphical depiction of the number in the queue at the ice cream store is in Figure 1.3.



**Figure 1.3: Graph of the Number in Queue in the Ice Cream Store**

Consider computing the average number in the queue. Suppose we observed the number in the queue to be two one time and ten another time. Would you say that on an average number, we would expect to see six in line? Of course “not”! You need to know how long the value of two was observed and how long the value of ten was observed. Suppose we observe two in the queue for ten minutes of the time and ten in the queue for only one minute. So the queue was observed for a total of 11 minutes. As a result, the value of two was observed  $10/11$ ths of the total time while ten was observed  $1/11$ th of the total time. Our average then is  $2*(10/11) + 10 * (1/11)$  or = 2.73 customers. Another way of computing

is to realize that the total waiting time<sup>3</sup> observed was  $20 + 10$  over a total of 11 minutes, which also yielded 2.73 people. Now, looking at the data from the ice cream store, we need the percentage of time (40 minutes) that there was zero waiting, one waiting, two waiting, three, and so forth<sup>4</sup>. In our case, a maximum of two was observed, and the waiting time would be computed as seen in Table 1.5.

**Table 1.5: Calculating Average Number in Queue**

Number In Queue	Time Period	Time Spent (min)
0	0.00 – 9.98	$0*9.98 = 0.000$
1	9.98 – 13.5	$1*3.52 = 3.520$
2	13.50 – 19.36	$2*5.86 = 11.72$
1	19.36 – 23.07	$1*3.71 = 3.710$
2	23.07 – 27.22	$2*4.15 = 8.300$
1	27.22 – 33.82	$1*6.60 = 6.600$
0	33.82 – 38.18	$0*4.36 = 0.000$
1	38.18 – 40.00	$1*1.82 = 1.820$
<b>Total Waiting Time</b>		35.67 minutes
<b>Average Number in Queue</b>		35.67/40 = 0.89 Customers

- **Utilization:**

This performance is the percentage of time the server is busy servicing customers, which is calculated by dividing the time spent servicing customers by the total time available. For our example, the attendant is only idle during the time period between finishing servicing customer one and the arrival of customer two (i.e.,  $9.01 - 8.36 = .65$  minutes). Therefore, the attendant's utilization will be  $39.35/40$  or 98.4%:

- Other possible performance measures are maximum values, standard deviations, time between departures, etc.

**Table 1.6: Types of Performance Simulation Measures/Metrics**

Type	Description/Examples
Counts	The number of parts that exited, entered, etc. (e.g., production).
Observation-based Statistics	These measures typically deal with time, such as waiting time, time in the system, etc. SIMIO calls observation-based statistics “Tally” statistics.
Time-Persistence/Time-Average Statistics	These measures deal with numbers in the system, queue, etc., when the values can be classified into different states. SIMIO calls these “State” statistics.

*Question 4:* What is another example of an observations-based performance measure?

---

*Question 5:* What is another example of a time-persistence performance measure?

---

*Question 6:* Is average “inventory” a time-persistent or observation-based statistic?

---

*Question 7:* Is the amount of time that a job is late a time-persistent or observation-based statistic?

---

<sup>3</sup> Total waiting time is also the “area” under the curve in Figure 1.3.

<sup>4</sup> We don't include 0 since it makes no contribution to the total waiting time.

## Terminology of a Queuing System

Figure 1.4 shows some elements of a “common” queuing system with the normal terminology. We will generally refer to the arriving objects as “model entities” and the counter/attendant as a “server.” You can see the members of the queue and the customer in service. The input processes for this system are the arrival process and service processes. We will provide details on these inputs later.

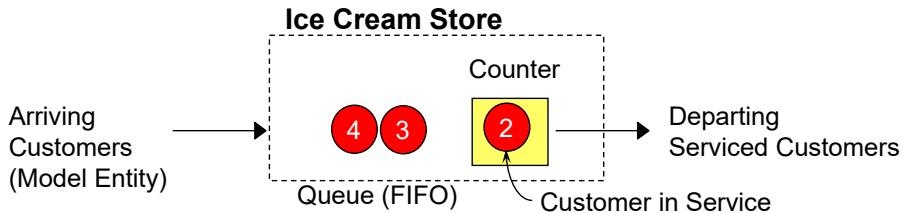


Figure 1.4: A Breakdown of Terminology

### Can we simulate it?

We want to reproduce the time-study data collection exercise we used for the ice cream store. But we want to synthesize it numerically as opposed to observing it – in essence, simulate it! We can compute the performance measures if we can recreate the time study data. Notice that the time study records are centered on “events.” Recall these events are points in time when the system changes its state (i.e., status). A quick review of that data reveals the following three events occur.

Table 1.7: Events of the Ice Cream Store

Number	Type	Event Description
1	Arrival	Indicates an entity will arrive at this time.
2	End of Service	An entity will be departing from service (i.e., service has finished)
3	End	The simulation will terminate and end all observations.

We need a method to keep track of our events to facilitate our simulation. Although there are other ways to keep track of events, keeping them in an “event calendar” is convenient. An event calendar contains the records of future events (i.e., things we think are going to happen in the future), ordered by time (with the earliest event first). Simulation can be executed by removing and inserting events into the event calendar.

Let’s add some specifics to our simulation problem (i.e., the ice cream store). We will use “minutes” as our base measure of time. The input data that we currently have is shown in Table 1.8.

Table 1.8: Input Data

Customer	Arrival Time	Interarrival Time	Processing Time
1	0.00	0.00	8.36
2	9.01	9.01	10.35
3	9.98	0.97	7.86
4	13.50	3.52	6.60
5	23.07	9.57	8.63
6	38.18	15.11	10.33
7	42.08	3.90	10.46
8	48.80	6.72	7.96

In the table, the arrival time has been re-stated as an “interarrival time” – namely time between arrivals. This restatement does not change the arrival times but simply how they are presented. Such a representation also requires the time of the first arrival from which the interarrival times are sequentially computed. Table 1.8 provides data on only the first customers. We may need more data, but this will be discussed later.

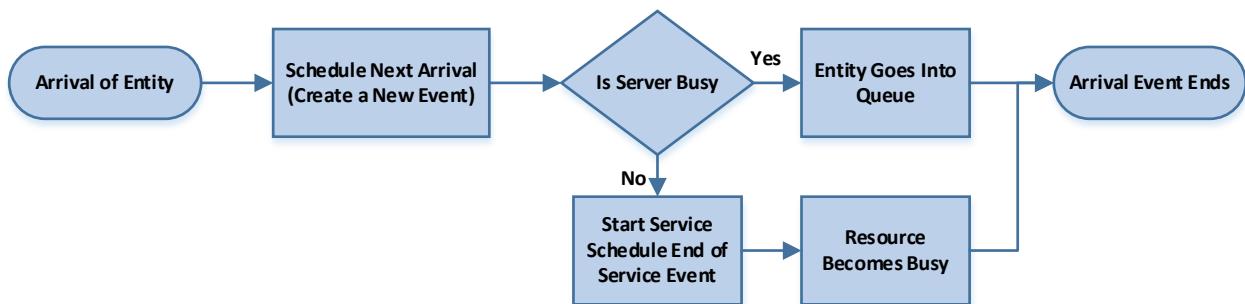
### The Simulation Algorithm

To synthesize the system as presented in the time study, we need a systematic means of moving through time by removing and inserting events on the event calendar. Consider the simulation algorithm shown in Figure 1.5.

- Algorithm Step 1:* Setup the simulation (i.e., initialize the system)
- Algorithm Step 2:* Remove the next event from the event calendar
- Algorithm Step 3:* Update simulation time (i.e., *TimeNow*) to the time of that event
- Algorithm Step 4:* Execute the processes associated with the event (i.e., adding additional events as needed and collecting statistics)
  - Arrival Event Process (see Figure 1.6)
  - Departure Event Process (see Figure 1.7)
  - End Event
- Algorithm Step 5:* Repeat *Algorithm Steps 2-4* until complete

**Figure 1.5: Simple Simulation Algorithm**

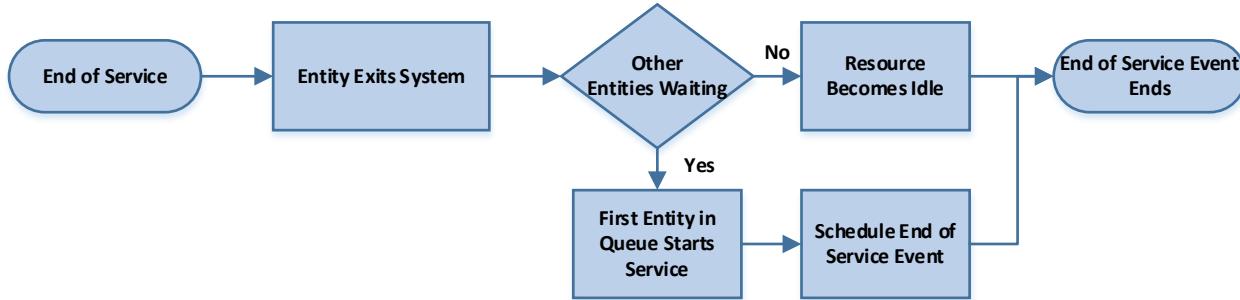
Remember that the event calendar is ordered according to the next most recent event. So, we can move through time by removing the “next” event from the event calendar, updating time to the time of that event, and executing whatever processes are associated with that event. This simple method is then repeated until we reach some terminating time or condition.<sup>5</sup> In our case, the “events” are the arrival of an entity, the service (departure) of the entity, and the end of the simulation period. The arrival and departure of the entities are the most important. Consider now how the arrival (see Figure 1.6) and the departure (see Figure 1.7) are processed within the context of our simple single queue, single server system. The word “schedule” means to insert this event into the event calendar.



**Figure 1.6: Event Process Associated with Arrival of Entity**

---

<sup>5</sup> The practice of removing the “next” event has caused some to refer to our simulation as a “next or discrete event simulation”.



**Figure 1.7: Event Process Associated with End of Service**

The arrival of an entity creates a new event because we know the next entity's arrival time since we have the interarrival times of entities. The newly arriving entity either must wait because the server (i.e., resource) is busy, or that entity can engage the server and start service. If the entity can start service, we now know a new future event, namely the service departure, because we know the processing time. So, suppose the removed event is an arrival. In that case, we insert the next arrival into the event calendar and may insert the service departure event in the event calendar, provided the entity can start service immediately. If the event removed from the event calendar is a service departure, then the entity that has finished service will exit the system. If the waiting queue is empty, then the resource (server) becomes idle. On the other hand, if at least one customer is in the queue, the first customer is brought into service, the resource remains busy, and a new future event, namely the service departure, is inserted into the event calendar because we have the processing time. Note that a service departure causes the entity to depart the system regardless of what happens to the server.

*Question 8:* What is the maximum number of new events that are added to the event calendar when an entity arrival event occurs? What are they?

---

*Question 9:* What is the minimum number of new events that are added to the event calendar when an entity arrival event occurs?

---

*Question 10:* What is the maximum number of new events that are added to the event calendar when an entity departure event occurs? What are they?

---

*Question 11:* What is the minimum number of new events that are added to the event calendar when an entity departure event occurs?

---

Finally, we note that the very first step of the simulation algorithm calls for the system to be “initialized.” In other words, how will we start the operation relative to the number of entities in line and the state of the server? It will be convenient to use the “empty and idle” configuration. By “empty and idle,” we mean that the server is idle and the system is empty of all entities.

### Part 1.3: Manual Simulation

A simple data structure will be employed to execute a manual simulation, as seen in Figure 1.8, to further understand how a discrete event simulation operates. Our manual simulation will consist of a “system animation” graphical representation of what is going on in the system, with the current customer being serviced shown inside the square and other customers waiting outside the square. The current simulation time (i.e., called “Time Now”) will be shown. The current event will be identified, followed by a description of the

process. Finally, the “Event Calendar” will be maintained, which will include the event time, event type, and entity ID number.

System Representation						
System Animation	TimeNow	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
□						

**Figure 1.8: Manual Simulation Data Structure**

**Step 1:** Algorithm Step 1 of the simulation, as defined in Figure 1.5, is used to “initialize” the system as seen in the executing structure of Figure 1.9. Note that we have inserted the time of the first entity arrival into the event calendar, and we have inserted the end event or the time to stop making observations.

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
□	0.0		Initialize simulation	0.0 40.0	1	Arr End

**Figure 1.9: System Initialization**

**Step 2:** Executing the simulation algorithm, we remove the next event (Algorithm Step 2) from the event calendar, update the simulation time to the time of that event (Algorithm Step 3), and execute the appropriate processes (Algorithm Step 4). The next event is the “Arrival of Entity #1” at time 0.0. The arrival of this entity allows us to schedule into the *Event Calendar* the “Arrival of Entity #2” to occur since the interarrival time between Entity #1 and Entity #2 is 9.01 minutes, and thus the *Arrival* event for Entity #2 is at time 9.01 (i.e., 0.0 + 9.01). Because Entity #1 arrives when the server is idle, that entity enters service, and we can now schedule its *Departure* event because we know the processing time for Entity #1 as 8.36 minutes; thus, the event time is 8.36 (i.e., 0.0 + 8.36). The result is that our data structure now appears in Figure 1.10.

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
1	0.0	Arr	Store opens and Entity 1 arrives and goes into service	8.36 9.01 40.0	1 2	Dep Arr End

**Figure 1.10: Processing Time 0.0 Event**

**Question 12:** We inserted the arrival of Entity #2 event before the departure of the Entity #1 event, so why is the departure event ahead of the arrival event in the event calendar?

**Question 13:** Why do we compute the new event times as  $0.0 +$ ? What does the 0.0 mean?

**Step 3:** We are done with the processes associated with time 0.0. The next event in the event calendar is the “Departure of Entity #1” from the system at time 8.36. Since there are no entities in the queue, the server is allowed to become idle, and no new events are added to the event calendar, as shown in Figure 1.11.

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
	8.36	Dep	Entity 1 departs service and server become idle	9.01 40.0	2	Arr End

Figure 1.11: At Time 8.36

**Step 4:** The next event is the arrival of Entity #2 at time 9.01. So, we remove it from the event calendar and update the simulation time to 9.01. We can schedule the arrival of Entity #3 at the current time (9.01) plus the interarrival time from Entity #2 to Entity #3 (0.97), which is event time 9.98. Next, Entity #2 can start service immediately since the server is idle, so its departure event can be scheduled as the current time (9.01) plus the processing time for Entity #2 of (10.35), which yields an event time of 19.36. The new status of the simulation is shown in Figure 1.12. We have also added an animation showing the status of the entities and the server.

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
	9.01	Arr	Entity 2 arrives and enters service	9.98 19.36 40.0	3 2	Arr Dep End

Figure 1.12: At Time 9.01

**Step 5:** The next event occurs at time 9.98, which is the arrival of Entity #3. Its arrival allows us to schedule the next arrival (i.e., Entity #4) at time 13.50. But now the arriving entity must wait, as indicated in the “System Animation” section of the data structure in Figure 1.13.

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
	9.98	Arr	Entity 3 arrives and must wait	13.5 19.36 40.0	4 2	Arr Dep End

Figure 1.13: At Time 9.98

*Question 14:* How did we get 13.50 for the arrival of Entity #4?

**Step 6:** The next event is the arrival of Entity #4, which will schedule the arrival of Entity #5 but will have no other actions since the server remains busy. The new simulation time is 13.50, and the updated status is shown in Figure 1.14

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
	13.50	Arr	Entity 4 arrives and must wait	19.36 23.07 40.0	2 5	Dep Arr End

Figure 1.14: At time 13.50

**Step 7:** Finally, we see that Entity #2 finishes service at 19.36 and departs. Entity #3 can go into service, and we can schedule the service departure of Entity #3 at 27.22, as shown in Figure 1.15

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
	19.36	Dep	Entity 2 departs and Entity 3 goes into service	23.07 27.22 40.0	5 3	Arr Dep End

**Figure 1.15: At Time 19.36**

*Question 15:* How did we compute the service departure for Entity #3 to be 27.22?

---

*Question 16:* What is the next event?

---

*Question 17:* What new events are added as a result of this event? (Give the event time, the entity, and the event type.)

---

**Step 8:** From Figure 1.15, we see the next event, which is the arrival of Entity #5 at time 23.07, which triggers the addition of the next arrival (Entity #6).

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
	23.07	Arr	Entity 5 arrives and must wait	27.22 38.18 40.0	3 6	Dep Arr End

**Figure 1.16: At Time 23.07**

**Step 9:** Entity #3 completes service at 27.22, allowing Entity #4 to enter serviced, which schedules the end-of-service event for Entity #4 at 33.82, as shown in Figure 1.17.

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
	27.22	Dep	Entity 3 departs and Entity 4 goes into service	33.82 38.18 40.0	4 6	Dep Arr End

**Figure 1.17: At Time 27.22**

*Question 18:* What is the next event?

---

*Question 19:* What new events are added as a result of this event? (Give the event time, the entity, and the event type)?

---

**Step 10:** The result of this next event is shown in Figure 1.18.

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
	33.82	Dep	Entity 4 departs and Entity 5 goes into service	38.18 40.0 42.46	6 5	Arr End Dep

Figure 1.18: At Time 33.82

Question 20: What is the next event?

---

Question 21: What new events are added as a result of this event? (Give the event time, the entity, and the event type.)

---

**Step 11:** We are getting closer to the time to quit observing the system; however, we have one more event and one more time update. The result of this next event is shown in Figure 1.18.

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
 	38.18	Arr	Entity 6 arrives and must wait	40.0 42.08 42.46	7 5	End Arr Dep

Figure 1.19: At Time 38.18

**Step 12:** Finally, the next event calls for the “End” of the simulation at time 40.0. Only time is updated as the event calendar is unchanged. The final state is given in Figure 1.20.

System Animation	Time Now	Event Type	Process Description	Event Calendar		
				Event Time	Entity #	Event Type
 	40.00	End	Simulation Ends	40.0 42.08 42.46	7 5	End Arr Dep

Figure 1.20: At Time 40.0

Our simulation approach is referred to as a “*Discrete-Event Simulation*” because the system only changes states at defined event times, and the state changes are discrete in that the number in the queue changes discretely, as well as the number in the system. If our model included variables that changed continuously with time, like the water in a tank, then we wouldn’t have a discrete-change system and would have to consider all points in time, not just those when the system changes state. Simulations that contain continuous variables are called “*Continuous Simulations*.” We can also have combinations of the two types of simulations. In fact, SIMIO can model both kinds of systems together – a multi-method simulation language.

## Part 1.4: Input Modeling and Simulation Output Analysis

Since we have now simulated the ice cream store time study, we can re-organize this information from an entity viewpoint and compute the *same performance statistics* that we computed previously!

*Question 22:* Do we have enough information to draw conclusions about the present system?

---

*Question 23:* What can we do to extend the information available?

---

If we had a total of 45 interarrival times and 43 processing times, we could simulate a total of 480 minutes of time. For example, we may obtain the information given in Table 1.9. We have included the minimum, maximum, and average values over the 480 minutes. Currently, we are simulating “only” one day for 480 minutes.

*Question 24:* Now, do you have enough information for a “present systems analysis?

---

*Question 25:* Is a one-day simulation long enough?

---

**Table 1.9: Final Performance Measures for 480 Minutes**

Performance Measure	Value
Total Production	43
Average waiting time in queue	9.59
Maximum waiting time in queue	35.65
Average total time in system	17.87
Maximum total time in system	42.65
Minimum total time in system	7.46
Time-average number of parts in queue	0.88
Maximum number of parts in queue	4
Ice Cream Attendant utilization	78%

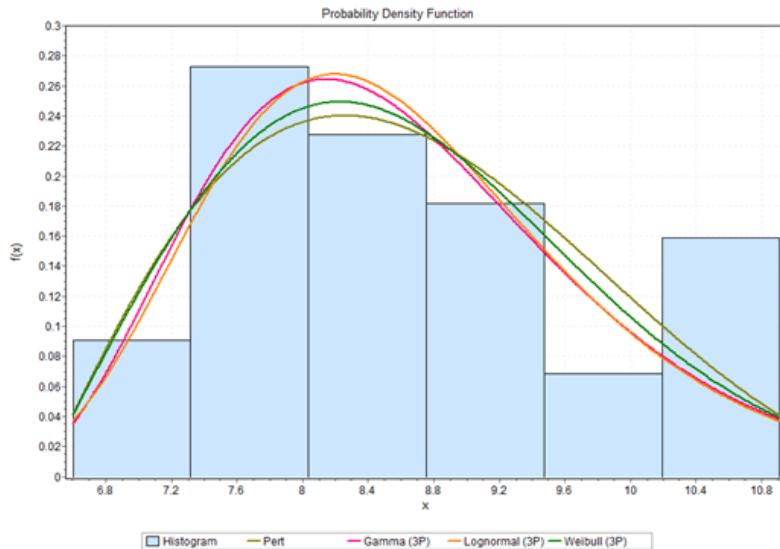
One day doesn’t give us much information about the day, so more “days” of information are needed. But that means more interarrival times and more processing times. If we did a simulation of ten days, we would need approximately 450 interarrival times and 430 processing times. If that information came from an electronic record, then getting more data may not be a problem.<sup>6</sup> However, if our only alternative is time studies, we will need to observe for ten days, which may be a greater intrusion into the actual operation than we expect.

An alternative to more data collection is for us to find a “model” of this input data (i.e., interarrival and processing times). Perhaps we can use a statistical representation since processing times and interarrival times are most certainly random variables. We can often match the processing times and interarrival times to standard statistical distributions. First, we make a histogram of the data we observed. Second, to match the data, we try to pick out a statistical distribution, like a Gamma, Lognormal, Weibull, or Pert. In fact, a wide range of software can help with this undertaking. An example of this activity is shown in Figure 1.21.

---

<sup>6</sup> An implication of this need for more data is that performance improvement should be one of the bases on which an information system is designed. The information should not be limited to accounting and reporting, but also performance improvement.

Statistic	Value
Sample Size	44
Range	4.31
Mean	8.6277
Variance	1.3242
Std. Deviation	1.1507
Coef. of Variation	0.13337
Std. Error	0.17348
Skewness	0.37355
Excess Kurtosis	-0.85959



**Figure 1.21: Matching Observed Processing Time to a Standard Statistical Distribution**

As a result of this input modeling, we can characterize a random processing or interarrival time by a statistical model (i.e., distribution). For example, we might use a Pert distribution which has three parameters: Pert (6, 8, 12) for a processing time or maybe a Gamma distribution which has two parameters: 6.3 + Gamma(3.1, 0.7) where the 6.3 is the “location” of the distribution’s origin.

Once we characterize an input with a statistical model, simulation technology allows us to “sample” from that statistical model. Repeated sampling will statistically reproduce the input model but, more importantly, give us an “infinite” supply of input data. With that extended data, we can simulate many days of operation. Having many days of simulation allows us to gain an estimate of the precision of the average performance measures we compute. For example, suppose we simulate our system for five days. Using simulation terminology, we performed five “replications” or “runs” of our simulation model. Each replication obtains new “samples” for its interarrival times and processing times. To compute the average and the standard deviation, we use the averages from each day (one day yields one average). The results are shown in Table 1.10.

**Table 1.10: Results from Five Replication of 480 Minutes Each**

Replication	1	2	3	4	5	Avg.	Std.Dev.
Avg. time in queue	9.59	15.26	12.98	8.08	21.42	13.47	5.26
Avg. no. in queue	0.88	1.62	1.17	0.76	2.31	1.35	0.63
Machine Utilization	0.78	0.86	0.75	0.78	0.81	0.80	0.04

*Question 26:* How many observations are used to compute the average and standard deviation in Table 1.10?

*Question 27:* Why only the averages instead of all the queue waiting times within a given replication?

We know that these statistics, like the average and the standard deviation, are themselves random variables. If we looked at another five days (either simulated or real), we wouldn’t get the exact same results because of the underlying variability in the model. We need to have some idea about how precise these summary statistics are. In order to judge how precise a given statistic is, we often use a *confidence interval*. For example, we computed the average waiting time as 13.47, but does this estimate have a lot of variability associated with it, or are we pretty confident about that value?

*Question 28:* Confidence intervals in statistics are based on what famous distribution as well as famous theorem?

---

To compute a valid confidence interval, we need to ensure that two assumptions are met. In particular, any observations for which we want to create a confidence interval must satisfy the assumption that the observations are “*independent and identically*” distributed. The other assumption is the data (i.e., observations) are normally distributed or can employ the **Central Limit Theorem (CLT)** be employed.

*Question 29:* Are the observations of the waiting times during a simulated day independent and identically distributed?

---

*Question 30:* For example, would the waiting time for entity #1, entity #2, and entity #3 be independent of each other? Would you expect the distributions of these waiting times to be identical?

---

*Question 31:* Would the number waiting in the queue at 9 am be independent of the number waiting at 8:45 am? Would the distribution of the number waiting be identical?

---

Almost no statistic (performance measure)<sup>7</sup> computed during a single simulation replication will be independent and identically distributed (e.g., the waiting time of entity #4 may depend on entity #3 waiting and processing times). So, instead, we use, for example, the average values computed over the day as an observation (i.e., we will determine the average of the averages).

*Question 32:* Are daily averages independent and identically distributed? Why?

---

Since these daily averages (or maximums or others) are independent and identically distributed, we can compute confidence intervals for our output statistics provided they are normally distributed or the CLT can be used.

### Confidence Intervals on Expectations

A confidence interval for an expectation is computed using the following formula:

$$\bar{X} \pm t_{n-1,1-\frac{\alpha}{2}} \frac{\hat{s}}{\sqrt{n}},$$

where  $\bar{X}$  is the sample mean,  $\hat{s}$  is the sample standard deviation of the data,  $t_{n-1,1-\frac{\alpha}{2}}$  is the upper  $1 - \frac{\alpha}{2}$  critical point from the Student’s  $t$  distribution with  $n - 1$  degrees of freedom and  $n$  is the number of observations (i.e., replications). Note,  $\frac{\hat{s}}{\sqrt{n}}$  is often referred to as the standard error or the standard deviation of the mean. As the number of replications is increased, the standard error decreases. So, the following calculation would compute a 95% confidence interval of the expected time in the queue using the data from Table 1.10.

$$13.47 \pm 2.776 \frac{5.26}{\sqrt{5}}, \text{ or } 13.47 \pm 5.53 \text{ minutes or the interval of } [6.93, 20.00] \text{ minutes}$$

---

<sup>7</sup> Of course there are exceptional cases.

*Question 33:* Are you confident in the 13.47 average?

---

*Question 34:* Would you bet your job that the “true” average waiting time is 13.47 minutes?

---

So, another way to express the confidence interval is given by the *Mean*  $\pm$  the *Half-width*, as in  $13.47 \pm 5.53$ , which SIMIO will use. The Central Limit Theorem allows the confidence interval calculation by asserting that the standard error (or the Standard Deviation of the Mean) can be computed by dividing the standard deviation of observations by the square root of the number of observations. If the observations are not independent and identically distributed, then that relationship doesn’t hold. As  $n$  increases sufficiently large, the CLT and the inferential statistics on the mean of the population become valid (i.e., the distribution of the means is Normally Distributed).

*Question 35:* Does the confidence interval we computed earlier (i.e.,  $13.47 \pm 5.53$ ) mean that 95% of the waiting times fall in this interval?<sup>8</sup>

---

*Question 36:* Does it mean that if we simulated 100 days that 95% of the average daily waiting times would fall in this interval?

---

*Question 37:* Or looked at another way. Can we say that there is a 95% chance that the “true, unknown” overall mean daily waiting time would fall in this interval?

---

So, the confidence interval measures our “confidence” about the computed performance measures. A confidence interval is a statement about the mean (i.e., the mean waiting time), not about observations (i.e., individual waiting times). Confidence intervals can be “wider” than one would like. However, in simulation, we control how many days (i.e., replications) we perform in our analysis, thus affecting the confidence in the performance measures.

*Question 38:* Using simulation, how can we improve the precision of our estimates (reduce the confidence interval width?)

---

So, we can run more replications in our simulation if we want “tighter” confidence intervals.<sup>9</sup>

## Comparing Alternative Scenarios

In the simulation, we usually refer to a single model as a simulation “scenario” or a simulation “experiment.” Our model of the present ice cream store is a single scenario. However, using simulation as a performance improvement tool, we are interested in simulating alternative models, such as our ice cream store. We would refer to each model as a scenario. So, for example, if we added a new ice cream-making machine, this change would constitute a different simulation scenario. If we started an ad campaign and expected an increase in the ice cream store business, we would have yet another scenario. In general, we would typically explore a whole bunch of scenarios, expecting to find improvements in the operations of the ice cream store. The simulation is our experimental lab.

---

<sup>8</sup> Prediction intervals are used for observations.

<sup>9</sup> Note that we can also create a smaller confidence level by using a larger significance level or  $\alpha$  value)

So, let's reconsider a different scenario for our ice cream store. What would happen if the arrival rate increased by 10% (i.e., more customers arrive per hour due to an ad campaign)? We could reduce the interarrival time by 10% and make five additional replications of the 480-minute day. The original and alternative scenario results are shown in Table 1.11, along with the confidence intervals.

**Table 1.11: Results of Original and Added Time Scenarios**

<b>Replication</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>Avg.</b>	<b>Std.Dev.</b>	<b>LCL</b>	<b>UCL</b>
Avg. time in queue	9.59	15.26	12.98	8.08	21.42	13.47	5.26	6.93	20.00
Avg. no. in queue	0.88	1.62	1.17	0.76	2.31	1.35	0.63	0.56	2.13
Machine Utilization	0.78	0.86	0.75	0.78	0.81	0.80	0.04	0.74	0.85

#### Increased Customer Arrival

<b>Replication</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>Avg.</b>	<b>Std.Dev.</b>	<b>LCL</b>	<b>UCL</b>
Avg. time in queue	14.77	24.42	18.39	12.10	31.18	20.17	7.69	10.62	29.72
Avg. no. in queue	1.50	2.77	1.80	1.18	3.74	2.20	1.05	0.90	3.50
Machine Utilization	0.85	0.92	0.82	0.82	0.85	0.85	0.04	0.80	0.90

Variability in the outcome creates problems for us. We recognize that we simply cannot compare scenarios of only one replication, but even with five, it's hard to know if there are any real differences (although it appears so). Once again, we need to rely on our statistical analysis to be sure we are drawing appropriate conclusions. When comparing different sets of statistics like this, we would resort to the Student's t-test. A way to conduct the t-test is to compare confidence intervals for each of the two scenarios. If the confidence intervals overlap, then we will fail to reject the null hypothesis that the mean time in the system for the original system equals the new system.

The 95% confidence interval for the original five days was [6.94, 20.00], while for the increased arrival rate, the confidence interval is [12.48, 29.76]. Now, comparing the confidence intervals, we see that they "overlap", meaning *we cannot say there is a statistical difference in the average waiting time*. Without a statistical difference, any statement about the practical difference<sup>10</sup> is without a statistical foundation.

*Question 39:* What can we do to increase our chance of obtaining a statistical difference?

---

*Question 40:* Is it necessary to have a statistical foundation for our recommendations?

---

Although it is probably unnecessary to have a statistical foundation for every recommendation, we strive to do so to avoid the embarrassing situation of making a claim that is later shown to be erroneous – especially since our job may be at risk. By striving to have a statistical foundation for our recommendations, we take advantage of our entire toolbox in decision-making and promote our professionalism.

---

<sup>10</sup> A practical difference means that the difference is important within the context of the problem. When we are concerned with a practical difference of unimportant or cheap items, then a statistical difference is not necessary.

## Part 1.5: Elements of the Simulation Study

The entire simulation study is composed of a number of elements, which we present here. Although these are presented in a sequential manner, rarely is a simulation study done without stopping to return to an earlier issue whose understanding has been enhanced. In many instances, we work on several of the elements at the same time. But regardless of the order, we usually try to complete all the elements.

- *Understand the system:* Getting to understand the system is perhaps the most intense step and one that you will return to often as you develop a better understanding of what needs to be done.
- *Be clear about the goals:* Try to avoid “feature or scope creep.” There is a tendency to continue to expand the goals well beyond what is reasonable. Without clear goals, the simulation effort wanders. During this phase, identify the performance measures of interest that will be used to evaluate the simulation.
- *Formulate the model representation:* Here, you are clearly formulating the structure and input for your model. Don’t spend much time collecting data at this point because, as you develop the model, its data needs will become clearer. Also, be sure to involve the stakeholders in your formulation to avoid missing important concerns.
- *Translate your conceptual representation into modeling software,* which in our case is SIMIO. A lot of time is spent learning SIMIO, so you have a wide range of simulation modeling tools with which to build this model and other models.
- *Determine the necessary input modeling:* At some point, data will need to be collected on the inputs identified during the formulation and translation of the system into a computer model. The initial simulation model can be used to determine which inputs are the most sensitive to the output that needs to be collected. Fitting distributions is generally better, but expert opinion can be used to get the model up and running.
- *Verify the simulation:* Be sure it is working as expected without errors. Do some “stress tests” to see if it behaves properly when resources are removed or when demand is increased. Explain any “zeros” that appear in the output. Don’t assume you are getting counter-intuitive results when they may just be wrong.
- *Validate the model:* How does the model fit the real world? Is the simulation giving sufficient behavior that you have confidence in its output? Can you validly use the model for performance improvement?
- *Design scenarios:* Determine which alternatives you think will improve the performance of the present system create the alternative simulation models, and associate the models with scenarios.
- *Make runs:* do the simulation experiments for the scenarios. Be sure your simulation output generates the appropriate performance measures. Make multiple runs for each scenario.
- *Analyze results and get insight:* Carefully examine the output from the scenarios and begin to develop judgments about how to increase the performance of the system. Be sure the statistical analysis supports your conclusions.
- *Make Recommendations and Document the Model:* Be sure to discuss the results with all the stakeholders and decision-makers. Make sure you have addressed the important problems and developed feasible recommendations. Document your work so you or someone else can return one year later and understand what you have done.

## Part 1.6: Commentary

If you have worked carefully through this chapter, you will have a fundamental understanding of simulation that is completely independent of the simulation software or any particular application. Some of the key points have been.

- A simulation model consists of a system structure and input.
- The insertion and removal of events drive a simulation.
- Random variables are used to represent input.
- Simulation statistics include observations and time-persistent values.

- A simulation may consist of many performance measures.
- Verification and validation are important concerns in any simulation.
- By using a confidence interval, we have some measure of variability as well as the central tendency of the simulation output.
- The computerization of simulation greatly facilitates its value in the present and in the future.

# Chapter 2

## Introduction to SIMIO: The Ice Cream Store

---

Simulation is a very useful tool. Just about everyone who learns about simulation gets excited about its widespread applicability. A simulation-based investigation can benefit practically any manufacturing, production, or service system. SIMIO can make the application of simulation easier to do and, at the same time, provide a powerful approach for addressing complex problems in designing and improving these types of systems.

Also, simulation is fun when you build visually appealing models, and SIMIO kicks the fun up a notch by offering 3D (i.e., three-dimensional) visualization. Not only does the animation allow you to see how the model is behaving, but people you work with can as well which will greatly increase your credibility with them.

### Part 2.1: Getting Started

We assume you have installed SIMIO on your computer and are ready to go.

**Step 1:** Invoke SIMIO by either opening the Start menu in Windows or by clicking on a SIMIO icon that perhaps you have located on the Windows taskbar or on the Desktop. The opening SIMIO window, as seen in Figure 2.1, has the standard Microsoft Office “look and feel.”

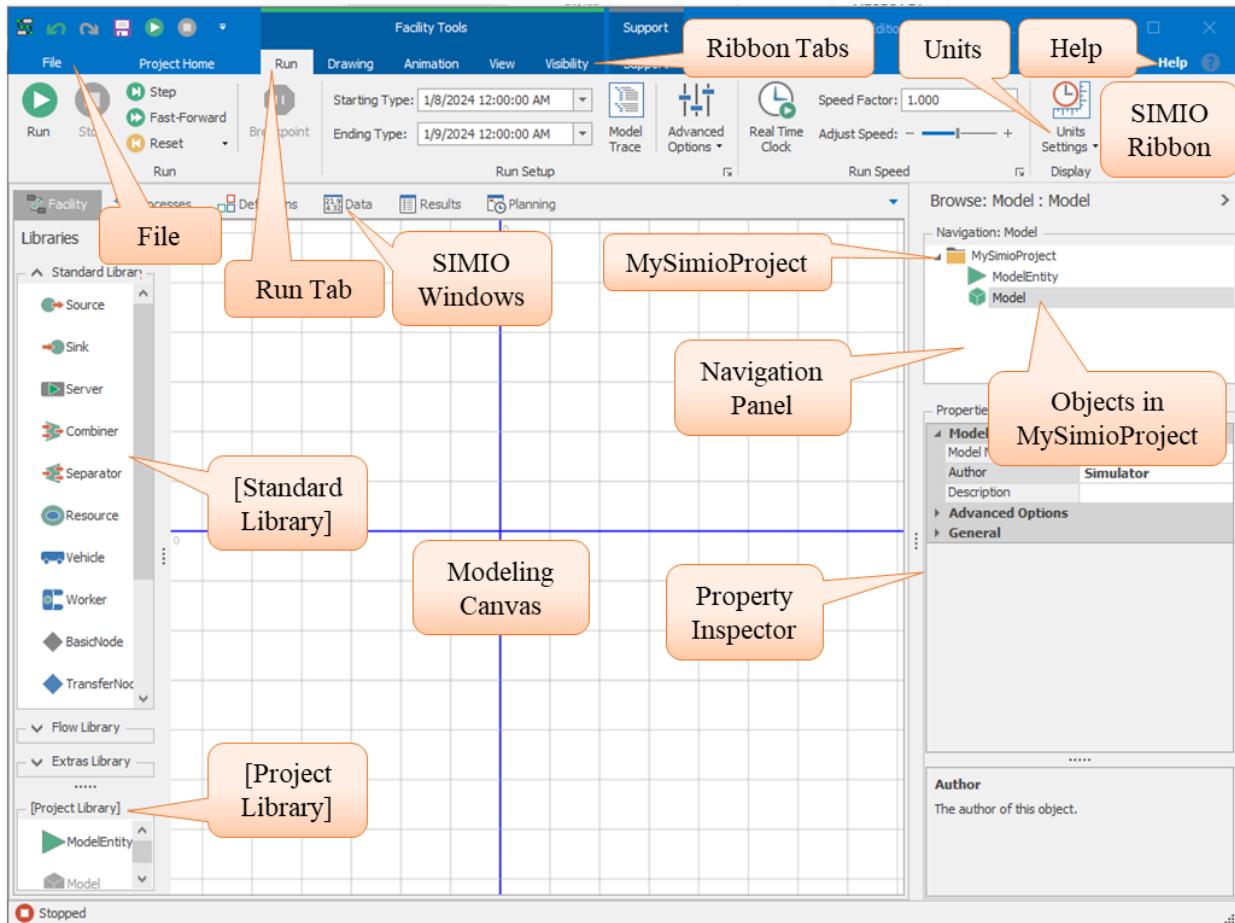
**Step 2:** When you open SIMIO for the first time, you will be placed into a modeling environment. The “Run Tab” of the “SIMIO ribbon” has been selected and contains the *Run*, *Run Setup*, *Animation Speed*, and *Display* sections.

**Step 3:** Below the SIMIO ribbon are the SIMIO window tabs. The “Facility” window has been selected as the canvas for building your simulation models. The SIMIO window displays the “Libraries” area on the left, a “Browse” section on the right, and the middle section, the modeling canvas.

**Step 4:** The *Libraries* section displays the [*Standard Library*], which contains the modeling object definition icons. Those definitions are used to create objects by clicking on the particular definition icon, dragging the icon to the modeling canvas to position it, and then clicking to complete the object definition. Most of the icons have names that connote their meaning. Also under the *Libraries* section is the [*Flow Library*], which will show the object definition icons for modeling various “flow” characteristics (i.e., continuous simulation). The [*Extras Library*] has additional specific objects to model cranes, robots, and elevators. Finally, the last portion of the *Libraries* section is the “*Project Library*,” which will contain the objects that become a part of your SIMIO simulation project, including the `MODELENTITY` object.

**Step 5:** In the *Browse* section, the top section is the “Navigation” panel, which identifies the components of your simulation model. By default, a new project will be named “MySimioProject” and will consist of two defined objects, a “`MODELENTITY`” and a “`MODEL`.” The lower section of the navigation panel displays the “*Property Inspector*.” This panel will display the properties of the object selected in the navigation panel.

**Step 6:** Finally, notice the “Help” options in the upper right-hand corner, which can also be invoked through the **F1** key. There is a lot of information to be found in the help documents, especially after you have some experience with SIMIO concepts and features.



**Figure 2.1: Opening SIMIO Window**

Other SIMIO resources available to you are found under the SIMIO ribbon tab “Support.” In the “Learning Simo” section, the “Sample SimBit Solutions” will refer you to a library of elemental models (each illustrating a particular modeling structure inside of SIMIO), and the “Examples” section will refer you to a library of complete models that provide interesting applications. There are also various books, videos, training, and guides. Also, note the SIMIO “version.”<sup>11</sup> in the “My Software” section to keep your version up-to-date.

**Step 7:** Two objects are automatically defined when a new model is created (see Figure 2.1). The MODELENTITY will create entities that move through our model, while the MODEL will contain positioned objects and the flow paths of entities. In a sense, the entities will roam around the fixed objects.

**Step 8:** In the [Navigation] panel, right-click the mouse on the MODELENTITY and then on the MODEL objects and select “Properties.”

*Question 1:* What is the “default” Model Name of the MODELENTITY?

---

*Question 2:* What is the Object Type of the MODEL (look under the “Advanced Options”)?

---

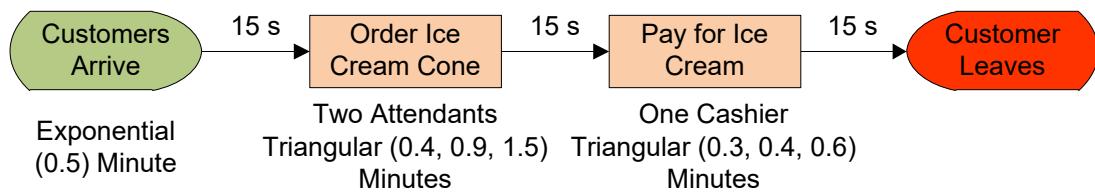
<sup>11</sup> SIMIO “Sprint” releases come out approximately monthly, so the software will be changing rather quickly. Be sure you obtain updates with new features and possibly bug fixes.

**Step 9:** Note the *Undo/Redo* buttons. These can be used to correct mistakes and recover previous modeling components. When an action cannot be undone, SIMIO will issue a warning.

**Step 10:** The “Units” are a convenient way to set the default specification units for time, length, rate, area, volume, and mass.<sup>12</sup>

## Part 2.2: The Ice Cream Store

A small ice cream store sells ice cream cones. Customers arrive and wait in line to be served by one of two attendants. These attendants take the ice cream order and give the cone(s) back to the customer, who then moves to the separate cashier to pay. After paying, the customers will leave the store. In building a simulation model, it is often important to flowchart the processes. Figure 2.2 shows the four processes of the ice cream store.



**Figure 2.2: Flowchart of Ice Cream Store**

In this problem, we will assume we know that:

- Customers arrive Exponentially with a mean interarrival time of 0.5 minutes,
- The time an attendant takes to interact with the customer and give them their ice cream cone(s) is modeled with a Triangular distribution with a minimum of 0.4 minutes, a most likely time of 0.9 minutes, and a maximum of 1.5 minutes,
- The time the cashier takes to accept payment for the ice cream is also Triangular with a minimum of 0.3 minutes, a most likely time of 0.4 minutes, and a maximum of 0.6 minutes and
- The travel time between each process is 15 seconds.

If both attendants are busy, customers will wait in a single line on a first-come, first-served basis. Likewise, there is a single waiting line for the cashier. We will also assume there is no limit to the length of the waiting lines.

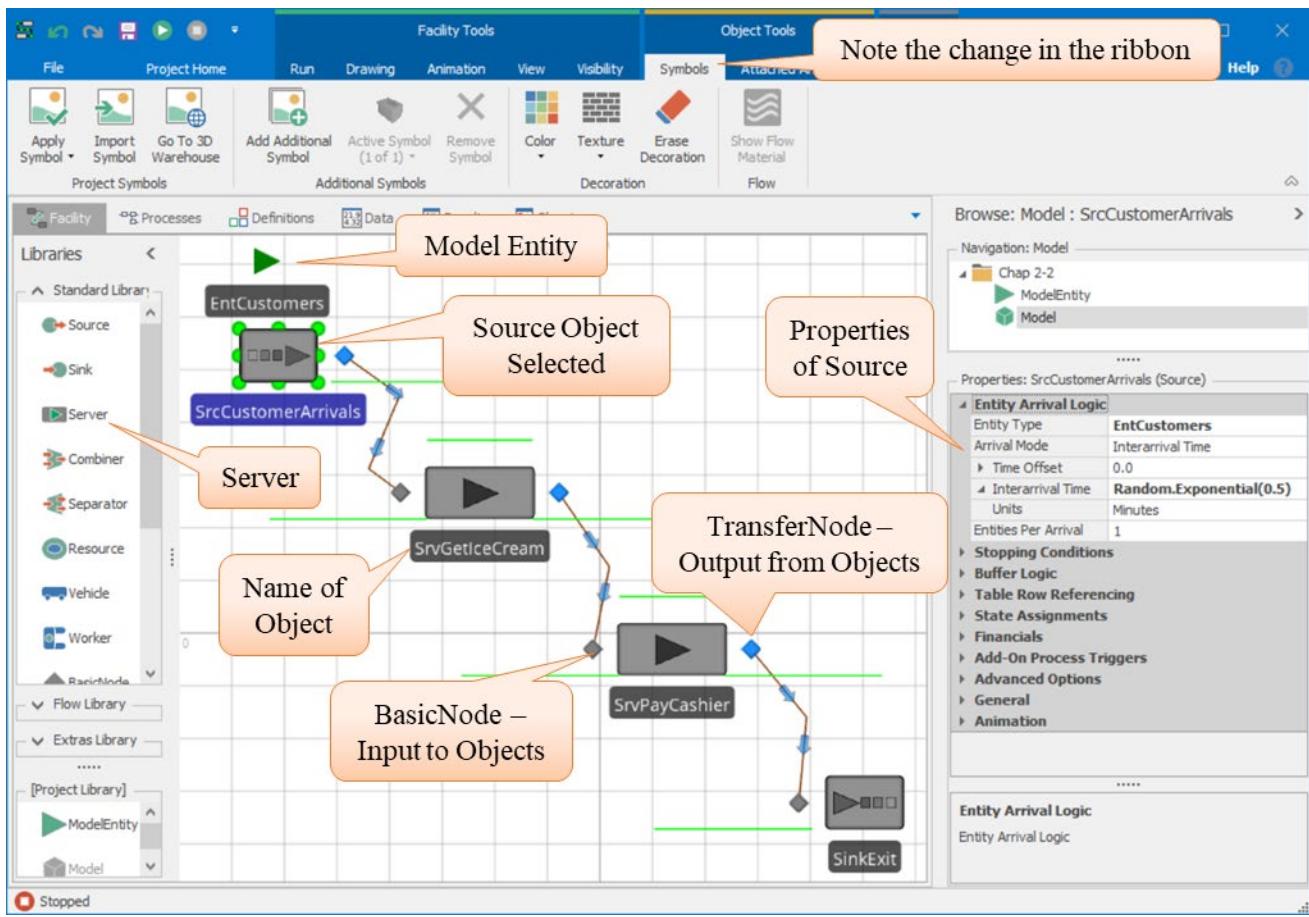
**Step 1:** It should be fairly clear that a `MODELENTITY` should model the customers while the `MODEL` will consist of the flowchart of the Ice Cream Store, as seen in Figure 2.2. Entities are the objects that will traverse through the network of fixed objects. Add an entity to your model by clicking on the `MODELENTITY` object in the *[Project Library]* panel and dragging it onto the canvas.

**Step 2:** Add objects to your model by clicking on the object type in the *[Standard Library]* panel. Drag the object around the modeling canvas and click to drop it into position (you can left-click and drag the objects around to relocate or delete them from the model). When you click on an object, its properties appear in the property inspector (typically on the bottom right side panel).

- Add a `SOURCE` object, two `SERVER` objects, and one `SINK` object as seen in Figure 2.3.
- Connect the process objects with `TIMEPATH` objects by clicking on the “output” node (blue diamond) of an object and connecting it to the “input” node (grey diamond) of an object. You can click between the nodes to produce a multi-segment path, which allows the path to be more flexible.

---

<sup>12</sup> SIMIO internally keeps all times in hours, distances in meters, rates in per hours, weights in kilograms, areas in square meters and volumes in cubic meters.



**Figure 2.3: Beginning to Model**

**Step 3:** Change the default “names” for each object. You can change the object’s name directly or in the “General” property section for each object. Use the names as specified in Figure 2.3.

**Question 3:** A SERVER object has three “lines” that surround it. What are they called? (note that a “.” adds further specification)

---

**Question 4:** A SERVER has an “input” and an “output” node. These nodes are also [Standard Library] objects. What is the standard library name of the input node?

---

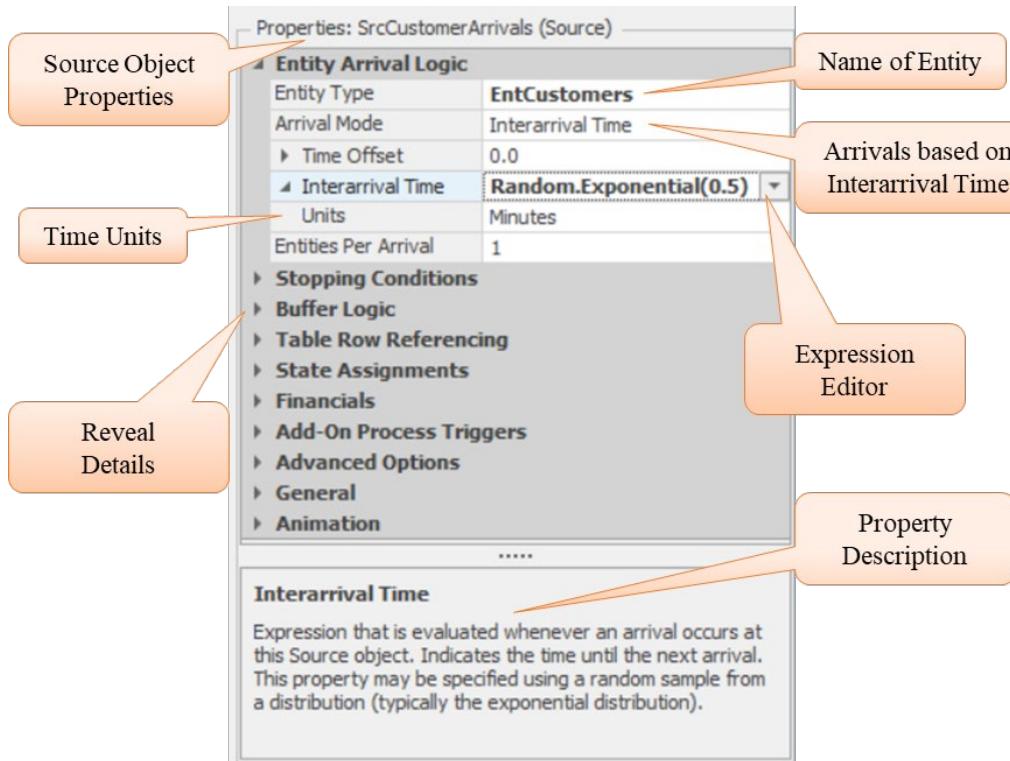
**Question 5:** What is the standard library name for the output node?

---

**Step 4:** Next, click on each object and fill out its properties according to our assumptions. Figure 2.4 shows the properties associated with the SOURCE object.<sup>13</sup>

---

<sup>13</sup> Note the default properties that are changed will be bolded to indicate a change has occurred.



**Figure 2.4: Properties of Source**

It is important to be sure the time units are correct for expressions involving time and distance. You may need to reveal the details in a dialog by clicking the icon to show the additional information. The *Expression Editor* allows you to write expressions that have the following form.

Object.SubObject.SubObject(parameters)

The expression editor begins by specifying a character. Then, select the object, its subobject(s), and its properties. The expression editor has “tab-completion,” which means the tab key will complete the name from its first few characters. It also indicates if subobjects are available for this object.

*Question 6:* What units of time are available for the interarrival time?

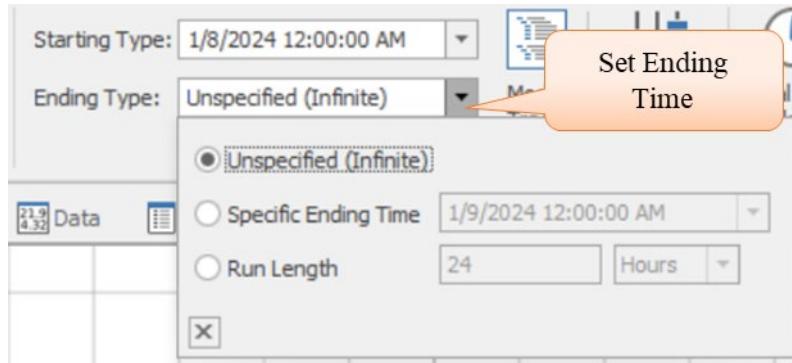
---

*Step 5:* Complete your modeling by adding the information for the two SERVERS and TIMEPATHS:

- For the **SrvGetIceCream** server object:
  - *Initial Capacity:* 2
  - *Processing time:* Random.Triangular(0.4, 0.9, 1.5)
  - *Units:* Minutes
- For the **SrvPayCashier** server object:
  - *Processing time:* Random.Triangular(0.3, 0.4, 0.6)
  - *Units:* Minutes
- For the **TIMEPATH** objects:
  - *TravelTime:* 15
  - *Units:* Seconds

Note that the object names are listed under the “General” category in the property inspector.

**Step 6:** Before we run the model, let’s change the ending time in the *Run Setup* section of the SIMIO “Run” tab to “Unspecified (Infinite),” as seen in Figure 2.5



**Figure 2.5: Setting the Length of the Run**

**Step 7:** Click the run button to start the simulation and let the simulation “run” for a while.

**Step 8:** Notice the MODELENTITIES () as they move through the model across the TIMEPATHS. The entities queue in the InputBuffer.Contents and then go through service in the Processing.Contents, are both shown as “green lines” that surround the SERVER objects. You can relocate the objects while the model is running.

**Step 9:** Click the “View” tab and select “3-D”.

**Question 7:** Hold down the left mouse button key and move the mouse left and right and up and down.  
What happens?

---

**Question 8:** Hold down the right mouse button key and move the mouse left and right and up and down.  
What happens?

---

**Step 10:** Switch between 2-D and 3-D using the “2” and “3” as “hotkeys”

**Question 9:** What happens in 2-D when you hold down the left mouse key and move the mouse left and right and up and down?

---

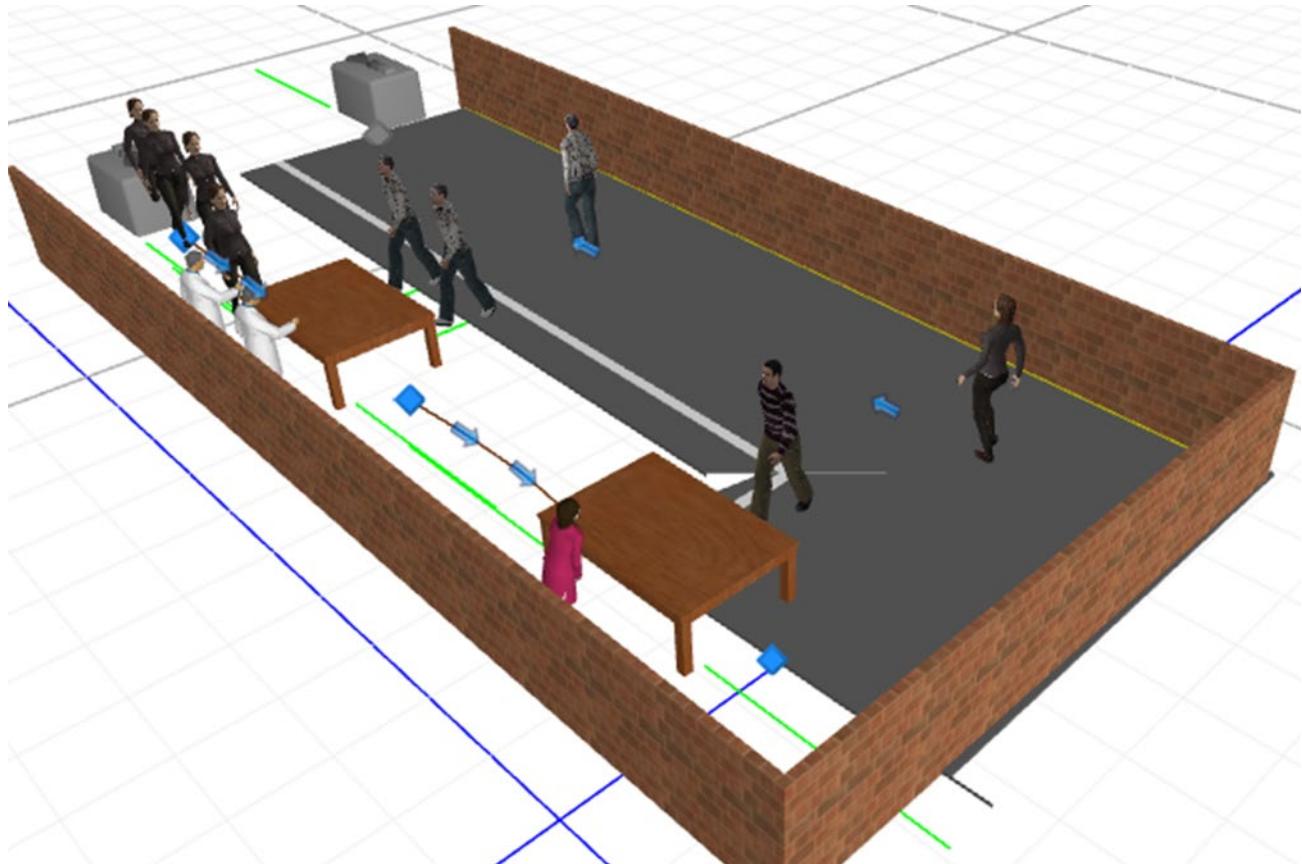
**Question 10:** What happens in 2-D when you hold down the right mouse button key and move the mouse left and right and up and down?

---

**Step 11:** Experiment in 2-D and 3-D by moving stations and changing the “layout” of the model. Change the travel time to the **SrvGetIceCream** station to 50 seconds to see another kind of change.

### Part 2.3: Enhancing the Animation

Animation is often what gets people interested in a model. Although a SIMIO 3-D model might take some work, it can bring a lot of attention. In this section, we will give you instructions on creating an animation that appears in Figure 2.6.



**Figure 2.6: The Animation**

**Step 1:** Make the ice cream store customers look like people rather than triangles. Select the **EntCustomer** entity (i.e., the green triangle), and under the *Symbol→Project Symbols* section, look at all the groupings and pick out a person from the “Library\People\Animated”, and click on it to substitute it for the triangle. Look at the picture in 3-D and enlarge it by pulling out one of the bounding ends. Run the simulation. You may want to change the 3-D perspective, as described previously.

*Question 11:* What happens to the animated people in the queues when you run the simulation?

---

**Step 2:** Change what happens to the animated people when they are waiting by clicking on the **EntCustomer** object and expanding the “*Animation*” details in its **Properties Inspector**. Change the “Default Animation Action” to “*Moving*” from “*MovingAndIdle*” to eliminate the shaking of the animated people when they are idle.

**Step 3:** Select the **SrvGetIceCream** SERVER object and substitute a “table” for the SERVER picture by choosing the table under the *Symbol→Project Symbols* section. Repeat the procedure for the **SrvPayCashier** SERVER object. Adjust the size of the SERVER objects to correspond to the persons’ size.

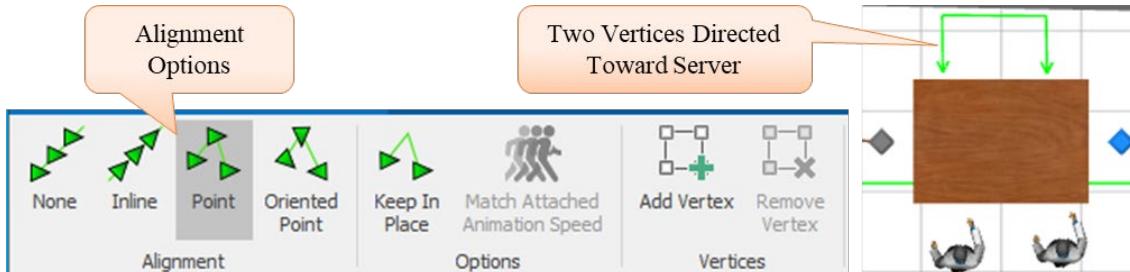
**Step 4:** Next, let's add some stationary people behind the tables to represent the attendants. Switch to 3-D and select the “Drawing” tab to do that. Click on the “Place Symbol” button and choose a person from the “Library\People” to represent the attendant. These non-animated libraries of people do not move their body parts if used as a symbol for an entity and move by “skating” from point to point. When the person is placed, you can click on one of the corners and hold down the *Ctrl* key to rotate the picture. Place the person behind the table. Duplicate the attendant by *Ctrl-C* and *Ctrl-V*. Now do the same for the **SrvPayCashier** at the cashier station, as seen in Figure 2.6.

**Step 5:** By default, the animated queues (i.e., the three green lines around the server objects) have the entities oriented in the same direction as the entities are traveling. Modify the orientation within the queue as specified in Table 2.1 by selecting the **Processing**.**Contents** queue and click the Point or Oriented Point button in the *Appearances→Alignment* section, as seen in Figure 2.7. You can use **Inline** for the **InputBuffer**.**Contents** queues.

**Table 2.1: Entity Alignment Options for Animated Queues**

Queue Alignment Option	Description
None	Entities will point from left to right regardless of the queue orientation within the object frame.
Inline	Entities will be oriented to point forward along the line.
Point	Entities only reside on vertices of the queue but will point in the same direction for all vertices.
Oriented Point	Entities only reside on vertices but can be aligned in different directions for each vertex. <sup>14</sup>

**Step 6:** Repeat the process for the animated queue for the processing contents for the **SrvPayCashier**.



**Figure 2.7: Changing the Orientation of Entities in the Animated Queues**

**Step 7:** Move the symbols around on the modeling canvas to better represent the ice cream store. You can add walls to your store using the “Polyline” tool from the *Drawing* tab. Be sure to set the object’s height in the “Object” section of the *Drawing* tab. The length measurements are in meters unless you specify that the units should be different. You may need to modify the heights of objects to make them consistent.

**Step 8:** The path from the **SrvPayCashier** object to the **SnkLeave** object can have a “path decorator” added. Path decorators may be added by clicking on the path and selecting a “Decorator.” To see people on this path, change the travel time to 300 seconds.

**Step 9:** Now, run the simulation and look at the animation. When you run the simulation, you may need to adjust the way it looks both in 2-D and in 3-D. The various queue symbols (lines) may need to be extended.

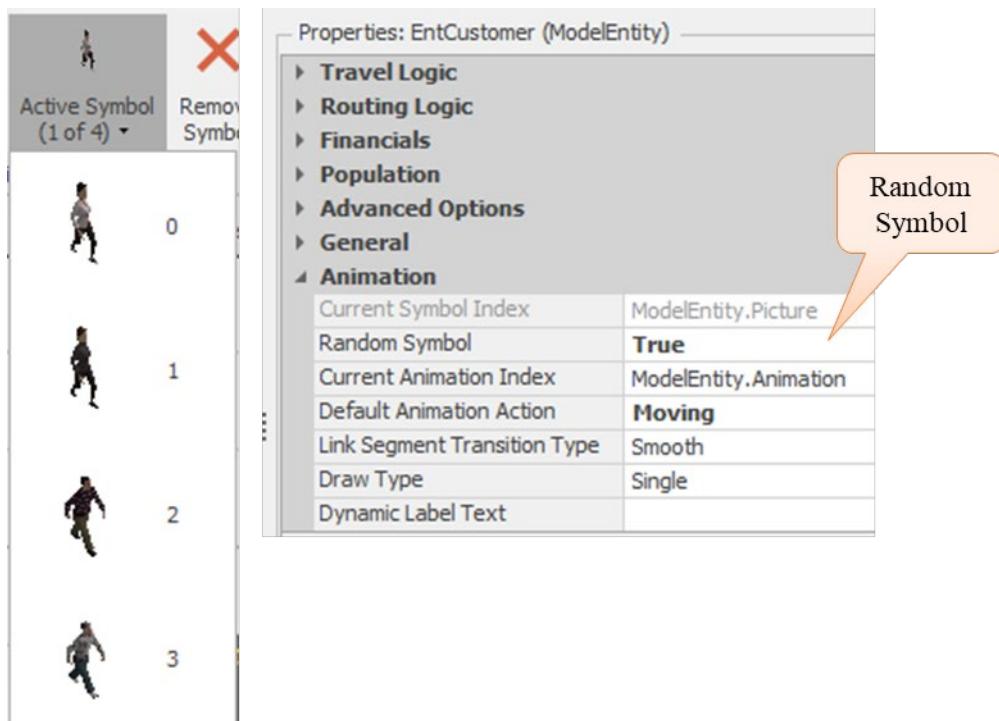
<sup>14</sup> Note the default queue alignment is None. Also, the **Keep in Place** option will force the entities to remain at the position they enter in the model. This characteristic is seen in queues that represent systems like waiting rooms (i.e., people do not change seats).

For instance, the line associated with the number of people waiting to get ice cream may be too short, so you might want to extend that line. Same is true for the `Processing.Contents` line. Remember that the length of the animated line does not impact the actual number in the line (i.e., there may be more entities waiting than can be displayed but just hidden).

*Question 12:* Show off your animation to your friends. What is their reaction?

---

**Step 10:** In the prior animation, the default entity triangle was changed to represent a person, but the exact same person arrived each time. To allow for different people to arrive, select the model entity and click the *Add Additional Symbol* button under the *Symbols*→*Additional Symbols* section to create as many types of people you would like to see arrive. Under the *Active Symbol* dropdown, select each symbol which will be currently identical to the first and change the symbol following the directions in *Step 1* as seen in Figure 2.8 where four total symbols have been added.<sup>15</sup> You will need to adjust the sizes of each of the symbols.



**Figure 2.8: Adding Additional Types of People**

**Step 11:** To use the new symbols, change the *Random Symbol* property to **True** under the *Animation* section of the model entity. Now, each time the Source creates a person, a random symbol will be selected as its picture. Run the new animation and observe what happens, as seen in Figure 2.6.

**Step 12:** In the [*Navigation*] panel, you will notice that your model has been updated with the new symbols.

## Part 2.4: Looking at the Results

While the animation brings attention to your model, you will build the simulation model to understand the numerical characteristics of the system being modeled. Typically, you will want to make all the changes to the

---

<sup>15</sup> The color of the clothing can be changed by selecting a color from the *Decoration* section and clicking on the part of the symbol you would like colored.

model until the model accurately represents the system and/or the potential changes have been identified before spending a lot of time on the animation.

**Step 1:** First, let's look at the results from the basic model, as illustrated in Figure 2.9. Perhaps we'll call this the "present system" model. Under the "Run" tab, change the "Run Length" to 8 hours and run the simulation. The results of this simulation are found under the "Results" tab. Using the "Fast-Forward" choice will not display the animation during the simulation. Whenever the simulation is run using the "Run" tab, it is run in "interactive" mode, allowing one to pause, step, etc., during the simulation, even extending the run length.

**Step 2:** Once you run the model, you can select the *Results* tab to access the statistics. Notice the "Unit Settings." Time is in hours by default, but it is easily changed using the *Unit Settings* button under the *Display* section. The time-based statistics will display the units in parentheses in the results.

**Step 3:** The results of a simulation are shown in the form of a "pivot table." A pivot table arranges the output data according to attributes that head each column, as seen in Figure 2.9.

Results Tab						
Drop Filter Fields Here <div style="float: right;">Filter</div>						
Output Attributes <div style="float: right;">Drop Column Fields Here</div>						
Average	Object Type	Object Name	Data Source	Category	Data Item	Statistic
						Average Total
ModelEntity	EntCustomer	[Population]	Content	NumberInSystem	Average	13.8041
					Maximum	26.0000
			FlowTime	TimeInSystem	Average (Hou...	0.1128
					Maximum (Ho...	0.2039
					Minimum (Hou...	0.0643
					Observations	964.0000
			Throughput	NumberCreated	Total	980.0000
				NumberDestroyed	Total	0.0000
Server	SrvGetIceCream	[Resource]	Capacity	ScheduledUtilization	Percent	2.0000
				UnitsAllocated	Total	2.0000
				UnitsScheduled	Average	1.9170
					Maximum	2.0000
				UnitsUtilized	Average	1.9170
			ResourceState	TimeProcessing	Average (Hou...	0.3739
					Occurrences	21.0000
					Percent	98.1367
					Total (Hours)	7.8509

**Figure 2.9: Looking at Results**

**Step 4:** Each Output attribute has two symbols – one to re-order the column and the other to "filter" or select the items displayed in the column.

**Question 13:** Filter the "Statistics" to show only the average values for the MODELENTITY. What is the average number and time in the system?

---

**Step 5:** Attributes can be moved left and right in the display.

**Step 6:** Right-clicking on the “Results Field” button (see Figure 2.9), you can display the “show field list.” You can drag other fields to display by dropping them in columns in the actual results.

**Step 7:** The “Categories” for statistics are Content, Throughput, Capacity, FlowTime, ResourceState and HoldingTime. These are applied to each data source as they are relevant.

**Step 8:** Resource utilization includes “ScheduledUtilization” as well as “UnitsAllocated,” “UnitsScheduled,” and “UnitsUtilized.” These items refer to the actual use of the object’s capacity and the average scheduled capacity. ScheduledUtilization is computed as a ratio of the actual time the resource is utilized divided by the total time its capacity is available. “UnitsUtilized” is the average number of units of the resource that are utilized up to the time this report is generated.

*Question 14:* What is the ScheduledUtilization of the capacity of the **SrvGetIceCream** object?

---

**Step 9:** Notice the waiting at the server object is displayed under the “HoldingTime” of the “InputBuffer” while the “Content” of the **InputBuffer** displays the number in the queue.<sup>16</sup>

*Question 15:* What is the average number (content) in the **InputBuffer** of the **SrvGetIceCream** server object?

---

*Question 16:* What is the average waiting time (holding time) in the **InputBuffer** of the **SrvGetIceCream** server object?

---

*Question 17:* Why does the Processing Content average number for the **SrvGetIceCream** server object equal the average Units Utilized previously?

---

*Question 18:* What is the utilization of the **SrvPayCashier** server object?

---

**Step 10:** To switch back to the model, select the “Facility” window tab, which is at the same level as the “Results” tab.

## Part 2.5: Commentary

If you have been paying close attention to the model construction, you may have noticed that:

- The SIMIO undo/redo is very convenient. This feature eases the task of trying different modeling features and then “undoing” and “redoing” various changes. Other features of the model, such as path type, can also be easily changed and then changed back by right-clicking on the link.
- You can suppress randomness in your simulation by selecting the “Advanced Options” within the *Run Setup* section of the “Run” tab and “Disable Randomness.” Doing this will allow you to follow the behavior without randomness to assist in debugging/verifying your model.

---

<sup>16</sup> The SIMIO nomenclature of “HoldingTime” and “Content” may seem non-standard. Generally, queuing theory uses the terms “waiting time” and “number in queue”. Also “waiting line length” or “number waiting” is used sometimes instead of “number in queue.”

If you are familiar with object-oriented design (OOD), the SIMIO [*Standard Library*] could be appropriately called a “class” library. The simulation objects are created from these class definitions. The act of selecting and dropping an object onto the modeling canvas is what “instantiates” the object. Properties, as displayed in the property inspector, define the characteristics of the objects. Later, you will see how to add your own characteristics. Properties are initialized but cannot be changed. Another type of characteristic, which is called “state variable” in SIMIO, may be added if the characteristic needs to be changed during the simulation. Generally, simulating with only one run (replication) does not generate reliable statistics for results. The next chapter will demonstrate how to obtain multiple replications for a given simulation scenario.

# Chapter 3

## Modeling Distance and Examining Inputs/Outputs

---

This chapter aims to model people arriving at an airport and going through the check-in process. In the first phase, we are only concerned with the amount of time it takes the passengers to reach the security checking station so we can determine the needed number of workers at a check-in station. Passengers arrive at the terminal and proceed to the check-in process to get their tickets. After check-in, the passengers proceed to the security checkpoint.

Passengers arrive according to an exponential distribution with an average of one arrival per minute. Passengers walk to the check-in station at a rate uniformly distributed between two and four miles per hour. Passengers travel 50 yards from the terminal entrance to the check-in station. Following check-in, they must walk 65 yards to the security checkpoint, as seen in Figure 3.1. The check-in station currently has four people to process customers who wait in a single line. The check-in process takes between two and five minutes, uniformly distributed. The simulation model needs to be run for 24 hours.

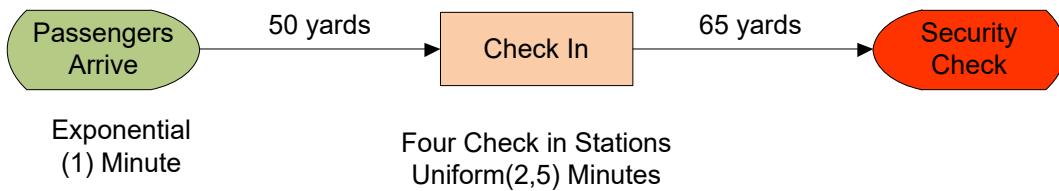


Figure 3.1: Airport Check-in Process

### Part 3.1: Building the Model

**Step 1:** Create a new model, as shown in Figure 3.2. Right-click on the objects in the [Navigation] panel and view their properties – you may need to right-click on the [Navigation] panel names and select their properties. Call the PROJECT “**AirportProblem**.”

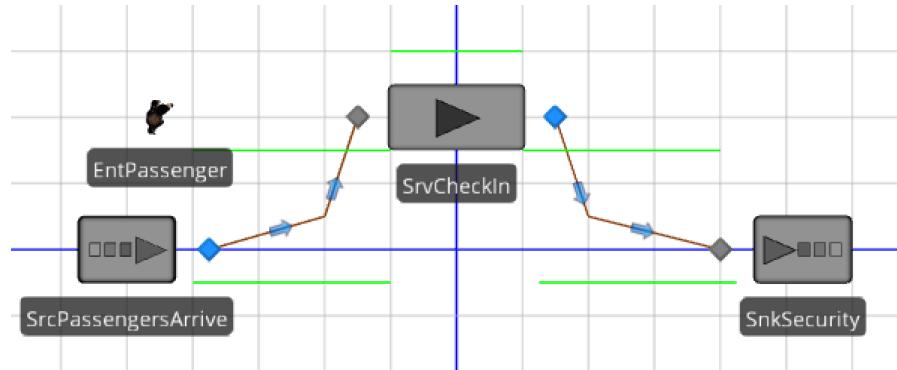
**Step 2:** Insert a SOURCE named **SrcPassengersArrive**, a SERVER named **SrvCheckIn**, and a SINK named **SnkSecurity** from the standard library to the Facility window and connect them using a path linkage (all of this can be done using the “Add-In” section on the “Project Home” tab).<sup>17</sup>

**Step 3:** Click and drag an entity from the MODELENTITY from the Facility window into the [Project Library] panel onto the Facility window. Also, change the name from **DefaultEntity** to **EntPassenger**. Change the passenger symbol to a person from the people library (either animated or static – we will use the static people pictures (“Woman1”) this time so you can see how they behave during the simulation).<sup>18</sup>

---

<sup>17</sup> To connect objects via links without selecting them from the [Standard Library], hold the *Ctrl Shift* keys down while selecting the TRANSFERNODE to start the drawing of the link. After completing the link, choose the correct link.

<sup>18</sup> You can also add more than one symbol for the passenger as was done in Chapter 2.



**Figure 3.2: The Basic Model**

**Step 4:** Set the speed at which a passenger can walk, namely, `Random.Uniform(2, 4)` miles per hour in the **EntPassenger** object. Set the *Interarrival Time* property, `Random.Exponential(1)` minutes in the **SrcPassengersArrive** SOURCE object. Set the *Processing Time* in the **SrvCheckIn** object to be `Random.Uniform(2, 5)` minutes and make sure to set its initial capacity to *four*.

**Step 5:** For the two paths, change the *Drawn to Scale* property of both paths to “*FALSE*” and set the correct distances for each. Recall that the path from arrival to check-in is 50 yards, while the path from check-in to security is 65 yards.

**Step 6:** Run the simulation and observe the behavior of the “passenger” and the changing color of the check-in.

*Question 1:* What does the passenger look like as they “move”?

---

*Question 2:* What color is the check-in when it is idle and when it is busy?

---

The symbol colors for the check-in can be seen by clicking on the *Active Symbol (1 of 9)* down-arrow in the *Additional Symbols* section. It shows the nine “default” states of the server.

*Question 3:* What is the state name and state number when the server is busy?

---

## Part 3.2: Using the 3D Warehouse

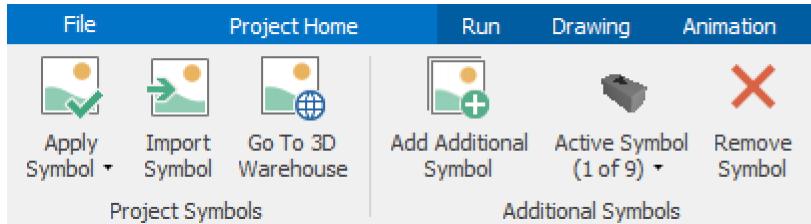
Although we have a working model, let’s change the server symbol. You can change the symbol of any entity or object by clicking the entity or object and changing it to some of the preloaded symbols in SIMIO. Alternatively, you can download new symbols from the **Trimble 3D Warehouse™**, or if you have symbols<sup>19</sup> already on your computer, you can import them into this model.

**Step 1:** To employ new symbols, select the **SrvCheckIn** object and click one of the nine choices among Project Symbols for the SERVER. The **Trimble 3D Warehouse™** should pop up when you click the “*Go to 3D Warehouse*” button, which will allow you to search for any symbol you want

---

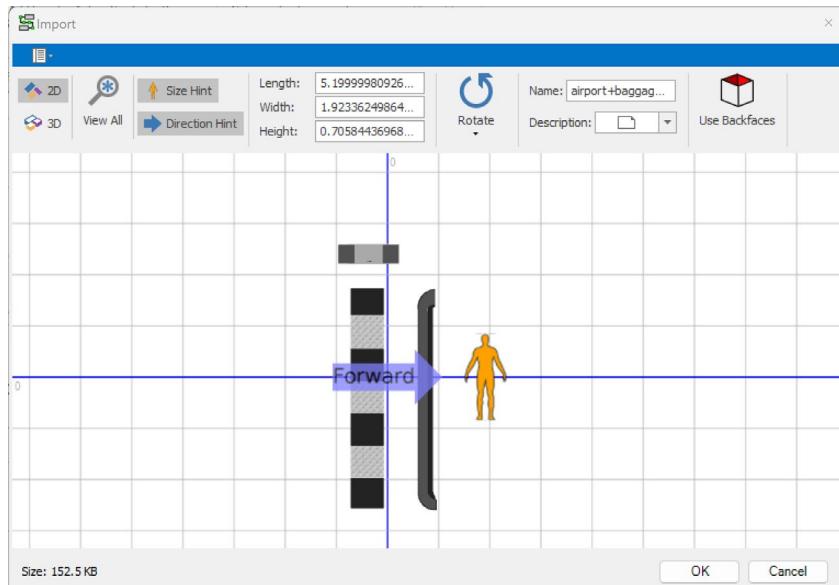
<sup>19</sup> New symbols can be created by packages like Trimble’s SketchUp Pro™ as well as other 3-D graphics software.

(e.g., try searching for “Airport Baggage” after selecting the *Models* tab). Once you find a suitable model, click the “Download” button on the top right, and the **Sketchup™** model will download.



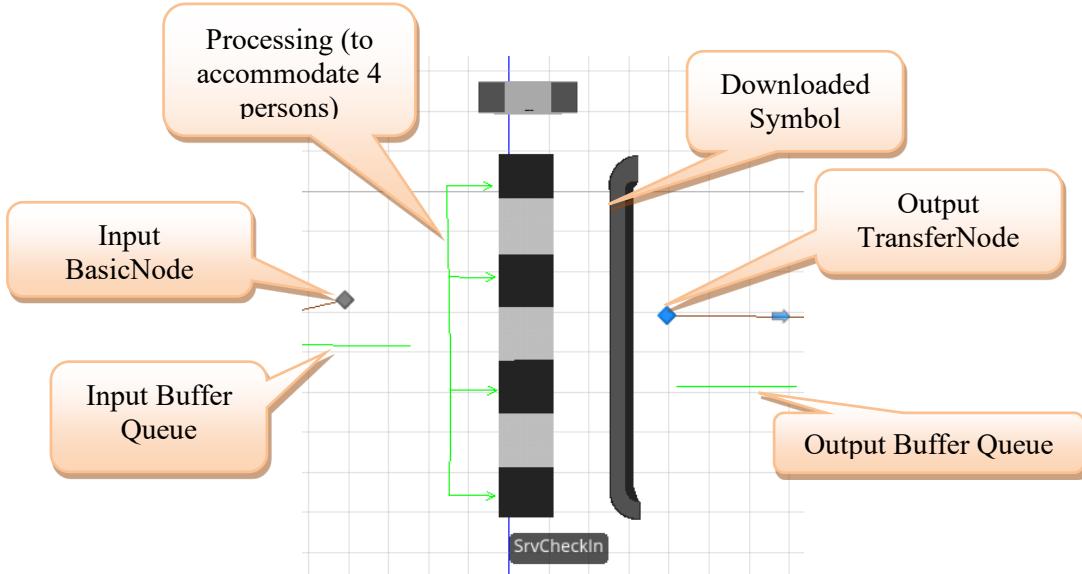
**Figure 3.3: Downloading and Importing 3D Symbols**

**Step 2:** Next, click the “*Import Symbol*”, and import the downloaded 3D Sketchup model. SIMIO will open an “Import” window where you can change some of its properties, such as size and orientation, before inserting the symbol into your model, as seen in Figure 3.3. First note: you can view the downloaded symbol in 2D and 3D. Also, it has length, width, and height that can be specified (in meters). A size “hint” is shown relative to a person so that you can modify the size based on that. Finally, you can rotate the symbol so it has the proper “orientation.” Change the *Length* to 5.2 m to make the symbol smaller and rotate the object by -90 degrees so the forward is the direction in which the entities will enter the object from the left.



**Figure 3.4: Import Trimble 3D Symbol Model**

**Step 3:** Position the baggage **SrvCheckIn** object so that the **Input BASICNODE**, the **InputBuffer** queue, and the **Processing** queue symbols give the appearance of check-in at an airport, as shown in Figure 3.5.



**Figure 3.5: Baggage Check-In**

**Step 4:** Run and adjust the animation (in 2D and 3D). Lengthen the **Processing** queue of the **SrvCheckIn** server and modify the symbol using “Oriented Point” to show that up to four people can simultaneously be in-process at the check-in station by adding two additional vertices, as seen in Figure 3.6.



**Figure 3.6: Modifying the Properties of the Queues**

### Part 3.3: Examining Model Input Parameters

Before looking more closely at the output, let’s re-examine our “input.” Using input parameters, we can use SIMIO to understand our assumptions about the interarrival time and processing time. Using input parameters instead of directly specifying the expressions allows one to use the same random expression across multiple objects (i.e., five machines all have the same processing time) but, more importantly, will enable response sensitivity and sample size error analysis to be performed.

**Step 5:** Click the “Data” tab and select the “*Input Parameters*” icon in the View panel on the left. Three types of input parameters can be defined: *Distribution*, *Table Value*, and *Expression*.<sup>20</sup>

**Step 6:** Click on “*Distribution*,” which means we are interested in a statistical distribution. By default, we are shown a histogram sample of 10,000 observations from a “Normal distribution” whose mean and standard deviation are one.

**Step 7:** Let’s try some other distributions. Look at a “Triangular” with a minimum of 0.4, a mode of 0.9, and a maximum of 1.5 (that was the “get ice cream” processing time from Chapter 1).

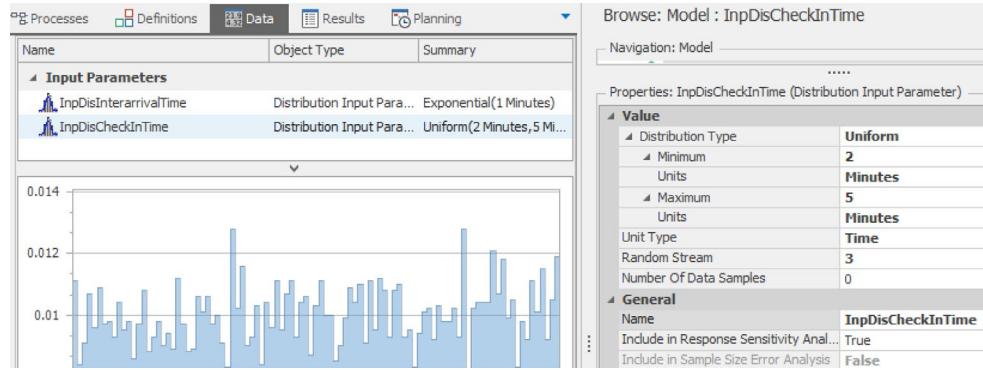
---

<sup>20</sup> *Distribution* allows one to specify a single distribution which can be used in both sensitivity and sample size error analysis by specifying the number of samples used to build the distribution.

*Question 4:* Using the SIMIO “help,” what is the mean of a Triangular distribution whose minimum is 1, whose mode is 3, and whose maximum is 8?

**Step 8:** Now define the interarrival time distribution of an Exponential with a mean of 1 minute (*Unit Type* is “Time” and the *Units* is “Minutes”) having with the name **InpDisInterarrivalTime**. It’s a good practice to give each distribution its own “Random Stream,” so we will use stream two.<sup>21</sup>

**Step 9:** Add a second distribution for the Check-In processing time. It is Uniform, with a minimum of two minutes and a maximum of five minutes. Name it **InpDisCheckInTime** and give it stream 3.



**Figure 3.7: Setting Up Input Distributions**

**Step 10:** Now go back to your model and replace (1) the Interarrival Time in the **SrcPassengersArrive** to **InpDisInterarrivalTime** and (2) the Processing Time in the **SrvCheckIn** to **InpDisCheckInTime**. The easy way to add these input parameters is to right-click the down arrow in the specification and select the “Set Input Parameter.” From the options, select the name distributions.

**Step 11:** Run the simulation model for a few minutes.

*Question 5:* Do you notice any change in the behavior of the simulation?

**Step 12:** You could continue using direct specifications like `Random.Exponential(1)`, or you can use the “*Expression*” choice in the “Input Parameters” to give `Random.Exponential(1)` a name to reference.

**Step 13:** Also, you should note that a given input parameter can be used in multiple places within an object, so changing all cases is reduced to changing the input parameter specification.

### Part 3.4: Examining Output

The specific output from a simulation is a random variable produced by the entities flowing through the model and receiving services. Their arrivals and services are both random. Hence, one run/replication of

<sup>21</sup> A random number stream for a distribution can be viewed as a specific series of random observations from that distribution. As such, it is reproduced by using that same stream again. This insures that the randomness associated with that stream is reproduced in other scenarios, so its randomness doesn’t add additional variance to the output performance measures.

the model produces one experimental value. This singular value is not precise, and to gain precision, we need to make multiple runs/replications.

**Step 1:** Change the run length to 24 hours and run the simulation (i.e., use “Fast-Forward”). After the simulation is complete, click “Stop” and then click the “*Results*” window tab to access the results window. In the “*Display*” section of the SIMIO ribbon, change the units of time to minutes.

*Question 6:* What is the average number of people waiting at check-in?

---

*Question 7:* What is the average time in minutes they were waiting?

---

*Question 8:* Do you think these are “good” results? Why or why not?

---

**Step 2:** To gain confidence in the precision of our results, the model needs to be run for several independent replications. From the “*Project Home*” tab, add a new experiment to run the model multiple times by clicking on the “New Experiment” icon.

**Step 3:** In the “*Design*” window tab, change the number of replications to ten<sup>22</sup> and “Run” the experiment using the icons in the “*Experiment*” section of the SIMIO ribbon. It should run quickly, and the progress of each replication is shown in the comment window at the bottom. If you are running on a multi-core computer, the runs will not be in order since each replication runs on a different core.

**Step 4:** You can look at the “*Pivot Grid*” or the “*Reports*” tab to get the results (remember that you can set the “units” in the ribbon for the output). From the *Pivot Grid*, notice that the “Results Fields” now include “Minimum,” “Maximum,” and “Half-width.” The half-width is one-half of a confidence interval with the “Confidence Level” as specified in the Experiment Properties, which is 95% by default. Recall that the confidence interval is a statement about the precision of the summary values.

*Question 9:* What did you get for the average, minimum, maximum, and half-width for the number of passengers in the system?

---

*Question 10:* What is the average, minimum, maximum, and half-width for the time in system (i.e., FlowTime) in minutes?

---

**Step 5:** With  $(1 - \alpha)\%$  as the “confidence level,” you may recall that the following formula gives the corresponding statistical confidence interval.

$$\boxed{\bar{X} \pm t_{n-1,1-\alpha/2} \frac{s}{\sqrt{n}}}$$

$n$  = number of replications

$\bar{X}$  = sample mean

$s$  = sample standard deviation

$t_{n-1,1-\alpha/2}$  = critical value from  $t$  statistic

---

<sup>22</sup> It should have defaulted to ten.

**Step 6:** Note that:

- The statistical summary values, such as an average, are computed average values from each replication (one observation per replication). The confidence interval will increase in width as the confidence level goes from 90% to 95% to 99%, while the confidence interval gets smaller as the number of replications increases.
- The “half-width” is simply one-half the width of the confidence interval
- According to the following equation, a simulation run for  $n_0$  replications with a half-width of  $h_0$  will require (approximately)  $n$  total replications to obtain a target half-width  $h$ .

$$n \approx n_0 \frac{h_0^2}{h^2} \quad h_0 = \text{half width from observation of } n_0 \text{ replications}$$

*Question 11:* Suppose you want the average number in the system for Passengers to be within  $\pm 5\%$  of its mean. How many replications would be needed, given the previous half-width calculation?

---

*Question 12:* Suppose you want the maximum number in the system for Passengers to be within  $\pm 2$  passengers. How many replications would be needed?

---

**Step 7:** You can add the standard deviation (actually the standard deviation of the mean or standard error – not the standard deviation of observations within a replication) by right-clicking one of the *Filter Fields* and selecting “Show Field List.” Next, double-click on the *Std. Dev* field.

*Question 13:* What did you get for the “Std. Dev.” for the number of passengers in the system?

---

*Question 14:* What part of the confidence interval formulation has the standard error – it’s what SIMIO calls the standard deviation here?

---

**Step 8:** To remove a field, simply move it back to the *PivotGrid Field List*.

### Part 3.5: Using Experiments

Suppose we want to determine the effect a change in the capacity of the check-in station will have. We could simply change the capacity and rerun the simulation, looking at the results. A better way is to set up a SIMIO “experiment” containing alternative “scenarios” that can be directly compared.

**Step 1:** In the *Facility* view, click on the **SrvCheckIn** object. Right-click on the *Initial Capacity* property and select “*Set Referenced Property*.” Use “*Create New Property*” and give it the name **CheckInCapacity**. Doing this creates a new model “property,” which is listed in the “*Definitions*” tab of the model under the “Properties” listing.

**Step 2:** From the “*Project Home*” tab, create a “New Experiment” and view the “*Design*” Tab. Note that this is a second designated “experiment” in the [Navigation] panel, and our referenced property is now listed in the columns.

**Step 3:** Add Scenario rows, changing the **CheckInCapacity** to two, three, and four to create an experiment grid, as seen in Figure 3.8. You can add rows by clicking on the “\*” symbol in the first column. Essentially, you are defining three scenarios, each having a different check-in capacity.

	Scenario	Replications		Controls	
*	Name	Status	Required	Completed	CheckInCapacity
	ScenarioCap2	Idle	10	0 of 10	2
	ScenarioCap3	Idle	10	0 of 10	3
	ScenarioCap4	Idle	10	0 of 10	4

**Figure 3.8: Experiment Scenarios**

**Step 4:** Now run the simulation, performing ten replications of each scenario, which is completed very quickly.<sup>23</sup> SIMIO will parallel process the experiments on each core of a multi-core computer. Replications that finish turn green in parallel with the yellow scenarios that are currently being run.

**Step 5:** Notice that the *Pivot Grid* report now includes columns for each scenario.

**Question 15:** What is the minimum average time in the system for “ScenarioCap4”?

---

**Step 6:** The “Reports” tab is an excellent way to compare scenarios. In the reports tab, the scenarios are organized under each data item.

**Question 16:** What is the average and half-width for an average time in the system for each scenario?

---

**Step 7:** It is helpful to add “responses” to the experiment to aid in comparing scenarios. Simply click the “Add Response” button in the “Design” tab. Insert the following two responses with expressions that will look at the average time in the system by the passengers and the total number processed, specifying units of minutes for the time in the system, as seen in Figure 3.9.

- TimeInSystem, in Minutes: EntPassenger.Population.TimeInSystem.Average
- TotalNumberProcessed: EntPassenger.Population.NumberDestroyed

General	
Name	TimeInSystem
Display Name	TimeInSystem
Expression	EntPassenger.Population.TimeInSystem.Average
Unit Type	Time
Display Units	Minutes
Objective	None

**Figure 3.9: Specifying the Time in System Response**

**Step 8:** Reset and rerun the experiment, looking at the “Response Result” charts and the scenario grid (see Figure 3.10). To arrange the “tabs,” click a tab and drag it to the desired position on the display—note the positioning points. You can also right-click a tab to configure the display of results.

---

<sup>23</sup> Run the experiments under the “Design” tab in the SIMIO ribbon to employ the scenario controls. If you go back to the Facility window to run the simulation, then the controls employ their original specifications.

**Step 9:** The *Response Results* and the SMORE<sup>24</sup> plots contain graphical displays of various summary results from each scenario. Generally, the numerical values are hard to determine from just the graph, which will be discussed later. Still, the graph helps to show trends and variations across and within scenarios. The confidence interval on the mean, which is most often used, is displayed in the rust color. If confidence intervals between scenarios don't overlap, the scenarios are statistically different.

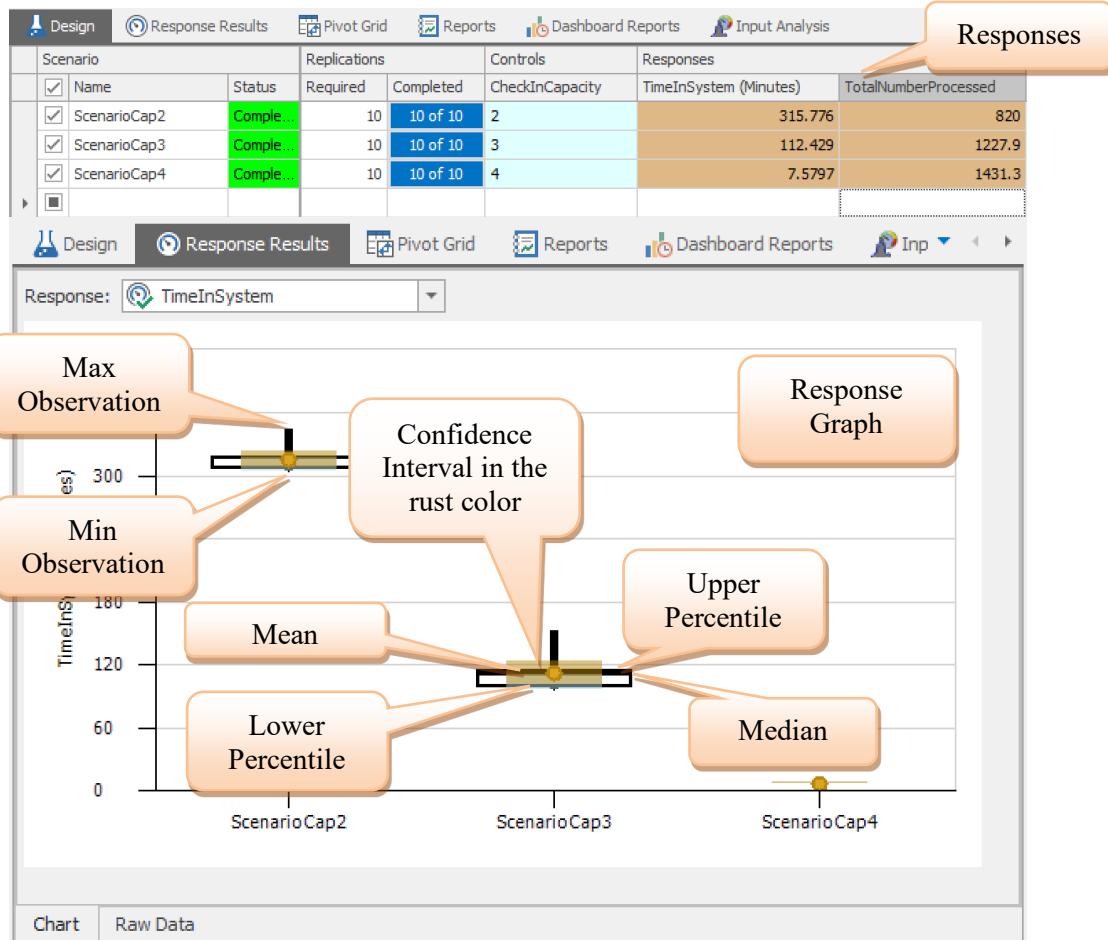


Figure 3.10: Response Results

**Step 10:** Clearly, as the capacity of the check-in increases, the time in the system decreases, and the number processed increases. You can modify the graph using the “View” section of the “Response Results” options in the ribbon. The response graph displays response values (see Figure 3.10) of observations within the simulation. The shading provides their confidence intervals, specified in the Experiment’s properties. Investigate clicking the “Histogram” and the “Means Line” buttons or try rotating the plot to get a different perspective.

**Step 11:** If you select the “Raw Data” tab found at the bottom of the “Response Results” tab, you will get the table shown in Figure 3.11, which includes numerical details on the responses, the confidence interval (95%) on the response, and the percentiles (which are 25% and 75% by default).<sup>25</sup>

<sup>24</sup> SMORE is SIMIO Measure of Risk & Error

<sup>25</sup> You can change the confidence level and percentiles in the EXPERIMENT properties.

Name	Values				Mean Confidence			Lower Percentile			Upper Percentile	
	Mean	Median	Minimum	Maximum	Half Width	Mean CI Be...	Mean CI End	Lower Value	Lower CI St...	Lower CI End	Upper Value	U
Scenario Na...	Response N...											
ScenarioCap2	TotalNumb...	820	822	808	833	5.40571201...	814.594287...	825.405712...	816	808	822	823
ScenarioCap2	TimeInSystem	5.26293011...	5.28019154...	5.07229868...	5.72312411...	0.13410455...	5.12882555...	5.39703467...	5.11506814...	5.07229868...	5.28019154...	5.29241345...
ScenarioCap3	TotalNumb...	1227.9	1228	1215	1241	6.92244275...	1220.97755...	1234.82244...	1221	1215	1228	1237
ScenarioCap3	TimeInSystem	1.87381966...	1.84764499...	1.60464073...	2.54320028...	0.19157668...	1.68224298...	2.06539634...	1.66678312...	1.60464073...	1.84764499...	1.91288972...
ScenarioCap4	TotalNumb...	1431.3	1423	1389	1516	26.1824803...	1405.11751...	1457.48248...	1412	1389	1423	1449
ScenarioCap4	TimeInSystem	0.12632830...	0.12113876...	0.10991814...	0.16195010...	0.01114226...	0.11518604...	0.13747056...	0.11873292...	0.10991814...	0.12113876...	0.12576010...

**Figure 3.11: Raw Data on Graph**

*Question 17:* What is the mean average time in the system for “ScenarioCap3”?

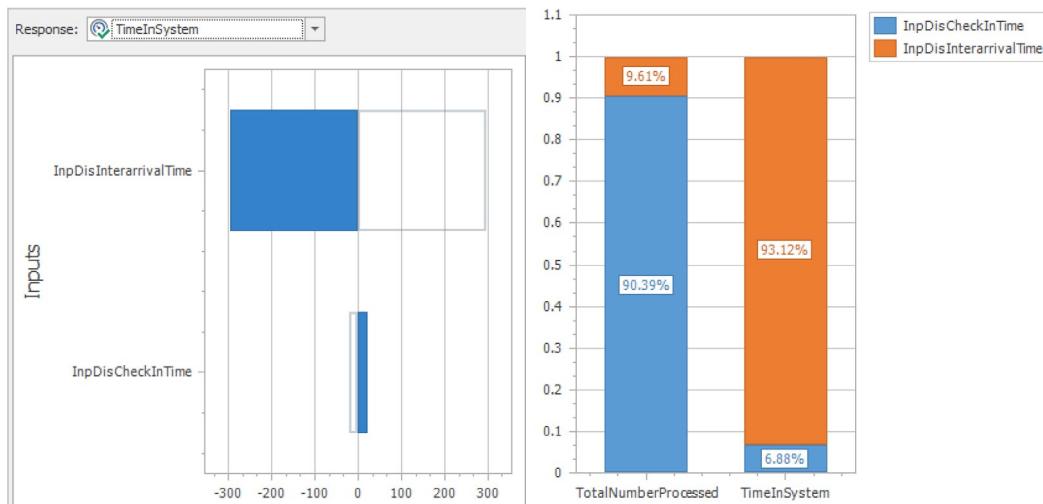
*Question 18:* What is the 95% confidence interval for the mean average time in system for “ScenarioCap3”?

### Part 3.6: Input Sensitivity

Since we specified the interarrival time and processing time as “input parameters” and our experiment has responses with multiple replications greater than the number of input parameters, SIMIO can perform a sensitivity analysis on these inputs as a by-product. Essentially, this analysis attempts to determine how the inputs affect the responses (i.e., outputs) through simple linear regression.

**Step 1:** Click on the “Input Analysis” window tab for the experiment and select the “Response Sensitivity” panel icon. Select, for example, “ScenarioCap2” and the “TimeInSystem” response.

**Step 2:** Four lower tabbed displays (i.e., *Tornado Chart*, *Bar Chart*, *Pie Chart*, and *Raw Data*) are available. Each chart measures how the particular “input” affects the particular “response” relative to each other. The numerical values associated with the graph are displayed by moving the mouse over the display. The Tornado chart shows a single response, whereas the bar and pie charts show all responses together, as seen in Figure 3.12.



**Figure 3.12: Tornado Chart for Sensitivity of Input to Time in System**

**Step 3:** The tornado chart in Figure 3.12 presents the results of the regression of the two inputs,  $InpDisInterarrivalTime(x_1)$  and  $InpDisCheckInTime(x_2)$ , on the output  $TimeInSystem(y)$ , which can be

represented more formally as  $y = \beta_1x_1 + \beta_2x_2$ . By passing your cursor over the bars in the tornado chart, you can see that  $\beta_1 = -291.261$  and  $\beta_2 = 18.993$ . These values have practical interpretation.  $TimeInSystem(y)$  is negatively related to  $InpDisInterarrivalTime(x_1)$  and positively associated with  $InpDisCheckInTime(x_2)$ . In other words, a unit increase in  $InpDisInterarrivalTime(x_1)$  causes the  $TimeInSystem(y)$  to decrease by 291.261 minutes, while a unit increase in  $InpDisCheckInTime(x_2)$  causes the  $TimeInSystem(y)$  to increase by 18.993 minutes.

**Step 4:** In the tornado chart, the solid line shows the relation between the input parameter and the response, while the outlined bar shows the negative. In this case, the lower the interarrival time (i.e., faster arrivals), the higher the time in the system. Meanwhile, the higher the processing time, the higher the time in the system. Clearly, from the figure, the interarrival time has a much more significant influence on time in the system than the processing time. However, from the bar chart, the reverse is true for the total number processed. This information can help modelers determine if more data needs to be collected to verify the input distribution assumptions.

**Question 19:** Passing your cursor over the bars, what are the sensitivity coefficients for the interarrival time and the processing time for ScenarioCap3?

---

**Question 20:** When you examine the Bar Chart and Pie Chart for the time in system response, what is the percentage contribution of each input for ScenarioCap3?

---

**Step 5:** Although the impact of adding capacity to the check-in is evident in Figure 3.6, the capacity can be considered an input parameter and added to the sensitivity displays.

### Part 3.7: Commentary

- Once you have a working model, you need to verify and validate your model. Is our model free of modeling and programming errors and working as intended (i.e., verify our model)? Does our model represent what is happening in the real world relatively well (i.e., validate the model)? We can never model exactly what is happening in the real world, but can we capture enough of the essence of the actual system to make a decision? For example, one can compare the actual values of a response to the confidence intervals created from the simulation experiment.
- The standard deviation produced in the output (see the “Results Field” option) is really the standard error (standard deviation of the mean). No standard deviation is computed within one run of the simulation, even though it can be computed and can provide some helpful modeling options (for instance, in establishing inventory policies). The danger is that the standard deviation calculated within a single replication cannot be used to compute a legitimate confidence interval since the observations may not be Normally distributed nor independent and identically distributed (IID).
- The sensitivity of input parameters on individual responses opens up a helpful method to explore the behavior of the simulation model, as seen in later chapters.
- Later chapters will describe the SMORE plots, subset selection, ranking/selection, and optimization in detail.
- While other simulation languages may manage the output in a database file, SIMIO manages the output internally, but it can be exported externally.

# Chapter 4

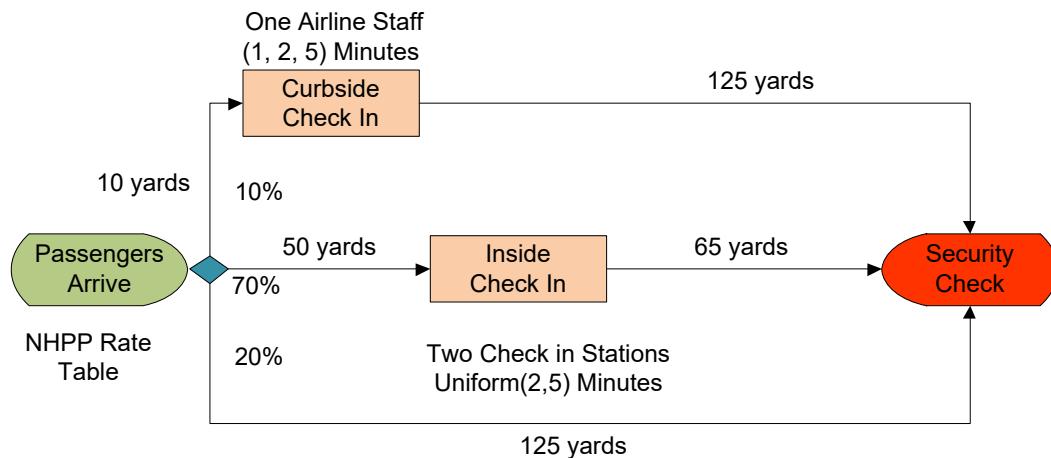
## More Detailed Modeling: Airport Revisited

There usually is more than one type of check-in in an airport and more than one type of passenger (i.e., a curbside check-in or a no-checked-bags check-in traveler). Our goal is to embellish the airport model from the last chapter to handle routing to different check-in stations, varying arrival rates of passengers, and varying types of passengers with rate tables and data tables. We will also introduce a means of giving entities characteristics, called state variables, which can be used to provide various distinctions.

### Part 4.1: Choice of Paths

As in the previous chapter, our primary concern is the amount of time it takes the passengers to reach the security checking station and the number of needed check-in stations.

**Step 1:** Delete the `EXPERIMENT` object in the prior chapter model by right-clicking on the Experiment in the [Navigation] Panel.<sup>26</sup> We will embellish the previous airport model to allow passengers arriving to choose three different routes to reach the security check. Passengers who need to check bags and get tickets can either check in at the curbside station (10%) or check in at the central station inside the airport (70%), while some passengers can proceed directly to the security checkpoint (20%). Refer to Figure 4.1, which depicts the modified airline check-in process to be modeled.



**Figure 4.1: Modified Airline Check-in Process**

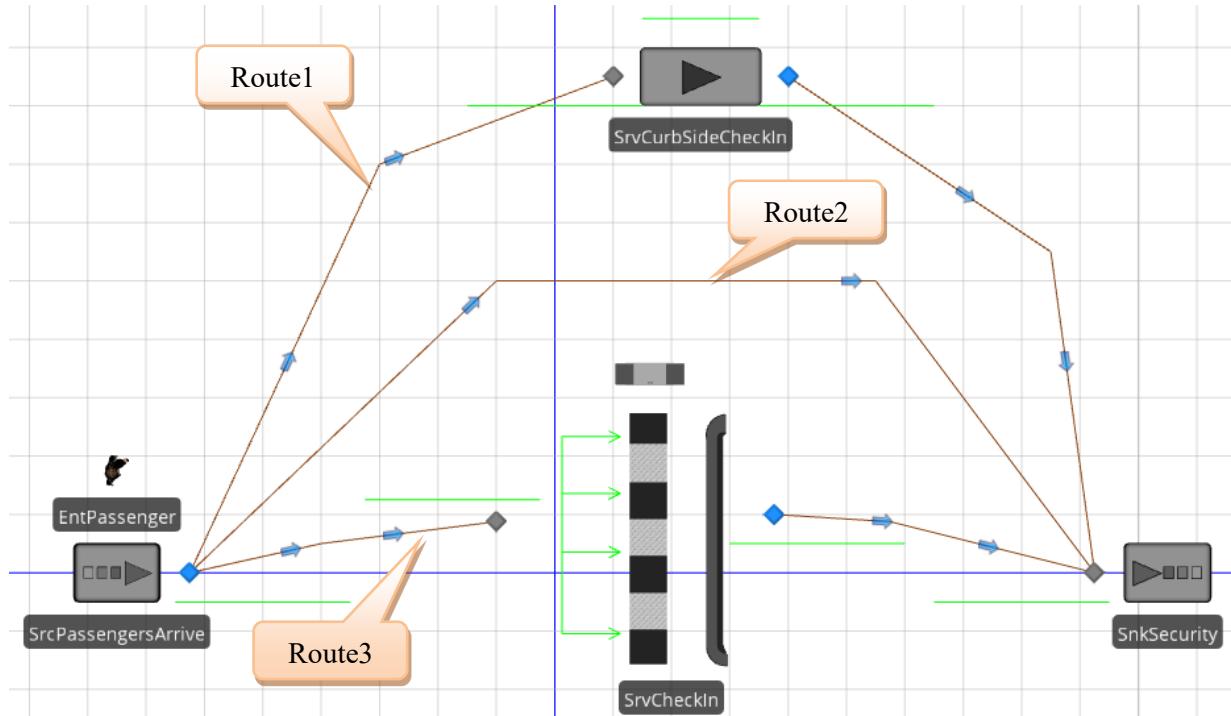
*Route 1:* Passengers who use the curbside check-in need to walk 10 yards to reach the station, where a single airline staff member takes between one and five minutes, with a most likely time of two minutes, to check-in a passenger. Once they have checked in, they walk 125 yards to the security check line.

*Route 2:* Passengers who have checked in online do not need to check any luggage; they only need to walk 125 yards to the security check line once they arrive at the airport.

<sup>26</sup> The [Navigation Panel] in the upper right corner contains all of the models, objects, and experiments in the project. You can access the properties, rename the item, or delete the item from the project.

*Route 3:* Passengers who plan to use the inside check-in first must walk 50 yards to reach the station where there are currently two airline personnel (instead of four previously) checking passengers into the airport. It takes between three and ten minutes uniformly to process each passenger. Once they have checked in, they walk 65 yards to the security check line.

**Step 2:** Modify the model from the last chapter to include the new curbside check-in (Server) and the direct path (i.e., Route 2) from the arrivals to the security gate, as seen in Figure 4.2. Specify the properties according to the description in Figure 4.1.



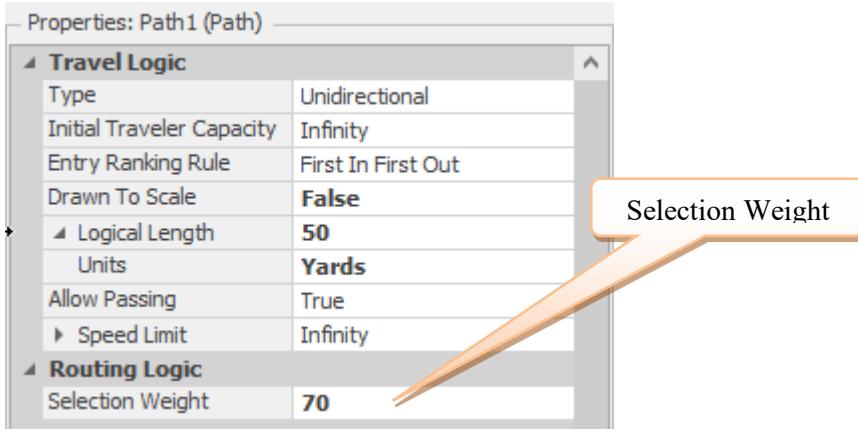
**Figure 4.2: Creating Different Routing Paths**

**Step 3:** It has been observed that passengers follow the probabilities in Table 4.1 in determining which route they will take to reach security.

**Table 4.1: Passenger Type Percentages**

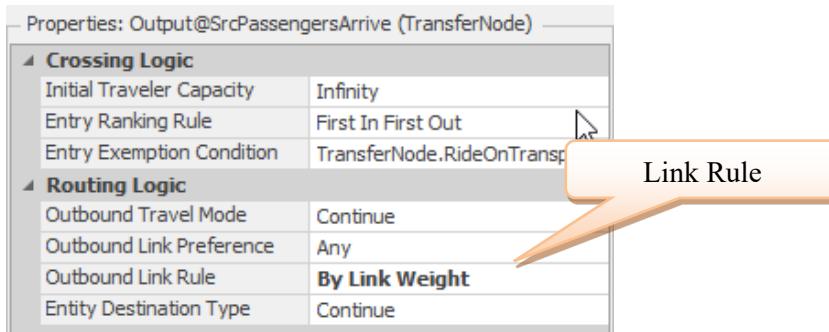
Check-in Type	Percentage
Curbside Check-in	10%
Inside Check-in	70%
No Check-in (Carry on)	20%

**Step 4:** To model the decision of a passenger in choosing their check-in route, the *Selection Weight* property of Path objects will be utilized. The probability of selecting a particular path is that the path's individual selection weight is divided by the sum of all path selection weights from that node object. Place the proper weights on each check-in route (see Figure 4.3).



**Figure 4.3: Weighting Paths**

**Step 5:** Remember to select the *Outbound Link Rule* to be “By Link Weight” on the TRANSERNODE **Output@SrcPassengersArrive** (see Figure 4.4).<sup>27</sup>



**Figure 4.4: Outbound Link Rule**

**Step 6:** Run the model for one 24-hour replication (i.e., no “Experiment” needed). You might want to improve on our animation.

*Question 1:* What is the average number of passengers waiting at Curbside and the Inside Check-in?

---

*Question 2:* What is the average wait time at each check-in?

---

*Question 3:* What is the average time it takes for a passenger to arrive and reach the security line?

---

## Part 4.2: Changing Arrival Rate

Passengers do not arrive at a constant rate (i.e., homogeneous process) throughout the entire day in an actual airport (even though they may arrive randomly), so we should expand our model to handle such phenomena.

---

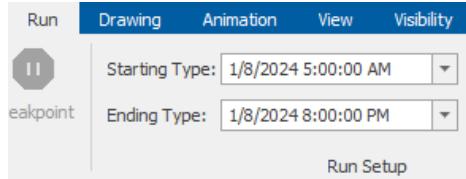
<sup>27</sup> Although “Shortest Path” is the default *Outbound Link Rule*, the “By Link Weight” rule will still be used unless the entity destination has been assigned to a particular node using by “Sequence, Specific, or Select from List” on the *Entity Destination Type*. Therefore, we could have used the default rule.

Our approach will be to allow the *arrival rates* to change throughout the day. To model this changing rate, we will use a data object in SIMIO called a *Rate Table* and reference the *Rate Table* we create in the arrival logic of our Source object. It is essential to understand that the interarrival times for this rate will be assumed to be Exponentially distributed, but the mean of this distribution changes with time. If an interarrival process has an exponential distribution, then it is statistically known that the number of arrivals per unit of time is Poisson distributed. Since the parameter (i.e., the mean) changes with time, it is formally referred to as a “non-homogeneous Poisson” arrival process, also called a NHPP (Non-Homogeneous Poisson Process). Therefore, when you use a SIMIO Rate Table, you assume the arrival process is an NHPP. Table 4.2 shows the hourly rate over the 24-hour day, where more passengers arrive on average during the morning and dinner hours. The zero rates imply no arrivals during that time period.

**Table 4.2: Hourly Arrival Rate During Each Hour**

Begin Time	End Time	Hourly Rate
Midnight	1 am	0
1 am	2 am	0
2 am	3 am	0
3 am	4 am	0
4 am	5 am	0
5 am	6 am	30
6 am	7 am	90
7 am	8 am	100
8 am	9 am	75
9 am	10 am	60
10 am	11 am	60
11 am	Noon	30
Noon	1 pm	30
1 pm	2 pm	30
2 pm	3 pm	60
3 pm	4 pm	60
4 pm	5 pm	75
5 pm	6 pm	100
6 pm	7 pm	90
7 pm	8 pm	30
8 pm	9 pm	0
9 pm	10 pm	0
10 pm	11 pm	0
11 pm	Midnight	0

**Step 1:** Arrivals are only present from 5 am to 8 pm for a total of 15 hours each day. A simulation run length of 24 hours would cause the time-based statistics like number in station, number in queue, and scheduled utilization to be computed when there were no arrivals, causing lower than expected values for these statistics. Therefore, we will set the *Starting Time* within the “Run Setup” tab to 5:00 am and the *Ending Type* to *Specific Ending Time* at 8:00 pm, as shown in Figure 4.5



**Figure 4.5: Replication Start and Stop Time**

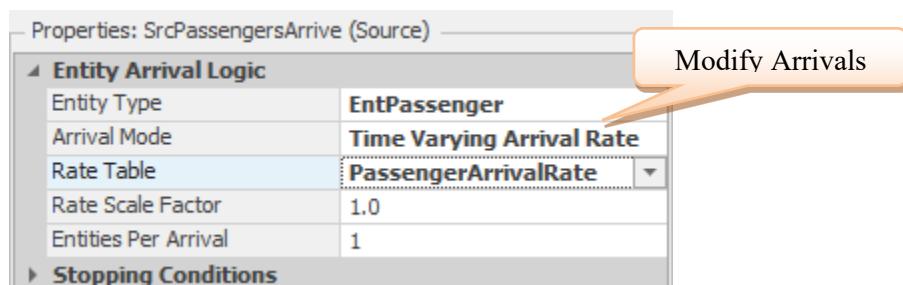
**Step 2:** Select the “Data” tab to create a Rate Table and then click *Create →Rate Table*. Name the table **PassengerArrivalRate**. Set the *Interval Size* to one hour and the *Number of Intervals* to 15, corresponding to the run length as seen in Figure 4.6. While you can modify the size (e.g., half-hour increments) and number of time intervals during which the arrival rate is constant, you cannot change the rate, which is fixed at *arrivals per hour*.

**Step 3:** Enter values in the rate table, as shown in Figure 4.6.

	Starting Offset	Ending Offset	Rate (events per hour)
▶	Day 1, 00:00:00	Day 1, 01:00:00	30
	Day 1, 01:00:00	Day 1, 02:00:00	90
	Day 1, 02:00:00	Day 1, 03:00:00	100
	Day 1, 03:00:00	Day 1, 04:00:00	75
	Day 1, 04:00:00	Day 1, 05:00:00	60
	Day 1, 05:00:00	Day 1, 06:00:00	60
	Day 1, 06:00:00	Day 1, 07:00:00	30
	Day 1, 07:00:00	Day 1, 08:00:00	30
	Day 1, 08:00:00	Day 1, 09:00:00	30
	Day 1, 09:00:00	Day 1, 10:00:00	60
	Day 1, 10:00:00	Day 1, 11:00:00	60
	Day 1, 11:00:00	Day 1, 12:00:00	75
	Day 1, 12:00:00	Day 1, 13:00:00	100
	Day 1, 13:00:00	Day 1, 14:00:00	90
	Day 1, 14:00:00	Day 1, 15:00:00	30

**Figure 4.6: The Rate Table (15 intervals)**

**Step 4:** Modify the **SrcPassengersArrive** source accordingly (see Figure 4.7 for details). *SIMIO refers to the NHPP as a “Time-Varying Arrival Rate” ARRIVAL MODE.*



**Figure 4.7: Specifying a Time-Varying Arrival Rate**

**Step 5:** Run the model and observe the results.

**Question 4:** What are the average wait times at each check-in location (Curbside and Inside)?

---

**Question 5:** What is the average check-in time for a passenger (Curbside and Inside)?

---

If the simulation length (i.e., replication/run length) exceeds the size of the rate table, then the rate table repeats until the simulation replication is terminated. So, if the run length should be longer and there should be no arrivals during that time, then the rate table should contain zeroes accordingly.

### Part 4.3: State Variables, Properties, and Data Tables

Properties and state variables can be characteristics of the “Model” or the “Model Entities.” A “property” of an object is part of its definition and is assigned when an object is created, but it can’t be changed during the simulation. Properties can be expressions, so their evaluation and creation time can yield different values. In contrast, a “state variable” is a characteristic of an object that can be changed dynamically during the simulation. Generally, when properties and state variables are associated with the model, they are global because their values are visible throughout the model. When properties and state variables are associated with entities, you need to access the model entity to access the particular entity’s state variable value.

In an actual airport, more than one type of passenger arrives. Some are continental travelers, while others are inter-continental travelers. Still, some passengers have handicaps and require special attention or are with families. The different types of passengers result in additional processing times. To model these three cases, we will use a combination of state variables and properties to store the information and reference the information when needed.

There is a 33% chance for each type of passenger to arrive at our airport. The processing times at the curbside and inside check-ins vary depending on the passenger type. Specifically, see Table 4.3 for the processing times.

**Table 4.3: Check-In Times for Different Passenger Types**

Passenger Type	Curbside Check-In Time (minutes)	Inside Check-In Time (minutes)
1	Triangular(1, 2, 5)	Uniform(2, 5)
2	Triangular(2, 3, 6)	Uniform(3, 5)
3	Triangular(3, 4, 7)	Uniform(4, 6)

The MODELENTITY will need to reference these data as specifications at the various check-in counters.

**Step 1:** Since the passenger type is not a general characteristic of SIMIO model entities, we need to find a SIMIO characteristic we can use to represent the type or define such characteristic ourselves. User-defined characteristics whose values can be re-assigned are the “STATE VARIABLES.” A state variable can be a characteristic of the “Model” (namely global) or a characteristic of the “ModelEntity.” Here, we are interested in the characteristics of a MODELENTITY.

**Step 2:** To create a MODELENTITY State Variable, select the “**ModelEntity**” object in the *Navigation* panel, as seen in Figure 4.8. Click the “States” panel icon in the “Definitions” tab. It is convenient to number our

passenger type, so select a DISCRETE “Integer” state from the ribbon. Let’s name it **EStaPassengerType**<sup>28</sup>. Next, we need to assign each MODELENTITY a value for passenger type.



**Figure 4.8: Selecting ModelEntity Object in the Navigation Panel**

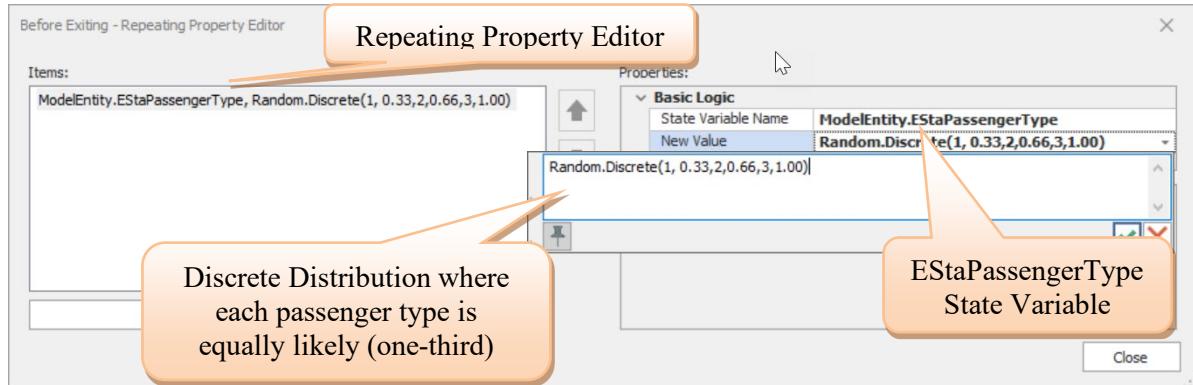
**Question 6:** When you define your new state variable, what other state variables are already defined for MODELENTITIES?

---

**Step 3:** We need to assign a value to **EStaPassengerType** for each entity. To obtain a random assignment of passenger type to entities (i.e., 33% chance for each type), we must use the Discrete Distribution (entered in cumulative distribution form).

`Random.Discrete(1, 0.33, 2, 0.66, 3, 1.00)`

**Step 4:** One convenient place is to make this assignment at the **SrcPassengersArrive** source object in the “State Assignments” property. Click on the plus box to reveal “*Before Exiting*.” Then click the ellipses box at the end of that specification to bring up the “**REPEATING PROPERTY EDITOR**,” which allows state value assignments in this case.



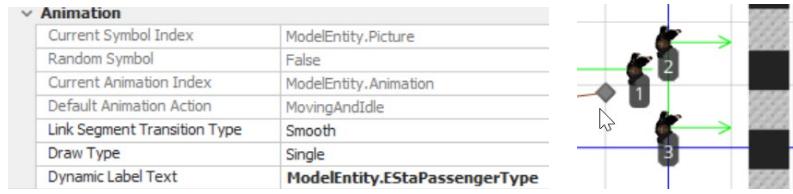
**Figure 4.9: Assigning Passenger Type**

Each passenger has a (randomly assigned) state variable that describes its passenger type.

**Step 5:** To verify that the model behaves the way one intended, we often add labels to display information while the simulation is running. Let’s add a dynamic label to the entity to display the passenger type, as seen in Figure 4.10.

---

<sup>28</sup> We will use “E” to denote “Entity” and “Sta” to denote “State variable”.



**Figure 4.10: Creating a Dynamic Label to Display the Passenger Type**

**Step 6:** We need to store the information for the check-in times. The expression of these times (e.g., `Random.Uniform(2, 5)`) does not change throughout the simulation. These check-in times need to be accessed by all entities. The check-in times are not specific to each entity but are characteristics of the model. So, “properties” are an appropriate way to store these data for the model.

**Step 7:** A way to organize model properties is through the use of a DATA TABLE. The data table is functionally identical to the table we created for . To create a DATA TABLE, click “Tables” in the “Data” tab, then “Add Data Table.” Call it **TablePassenger**. Characteristics, called “properties,” can be added to the data tables – these properties are the columns in the table. In our DATA TABLE, the processing times for each of the three types of passengers for each check-in location will be specified. Select the properties from the “Standard Property” option in the *Properties* section. **PassengerPriority** will be an *Integer* property, whereas the check-in times will be *Expression* properties named **CurbSideCheckInTime** and **InsideCheckInTime**.<sup>29</sup> Priorities 1, 2, and 3 represent Continental, Inter-Continental, and special needs passengers, respectively, as in Figure 4.11.<sup>30</sup>

Table Passenger			
	Passenger Priority	CurbSideCheckInTime (Minutes)	InsideCheckInTime (Minutes)
1	1	Random.Triangular(1, 2, 5)	Random.Uniform(2, 5)
2	2	Random.Triangular(2, 3, 6)	Random.Uniform(3, 5)
3	3	Random.Triangular(3, 4, 7)	Random.Uniform(4, 6)

**Figure 4.11: The Data Table**

**Step 8:** Since the check-in times are in minutes, we want to specify the “Unit Type” for these properties, as seen in Figure 4.12<sup>31</sup>.

<sup>29</sup> Note that if you first specify the “General” name, it will become the default for the “*DisplayName*” property. Also, by using proper case, the table labels as seen in the Figure 4.10 will have spaces between the words.

<sup>30</sup> Even though the check-in times are expression properties, the familiar drop down expression editor is not available in DATA TABLES. Therefore, the easiest way is to put in all three priorities and type in the first value of the first row. Then copy it to the other rows and just modifying the values of the parameters.

<sup>31</sup> It may seem obvious the Unit Type for the time properties should be specified. However, one can leave the units as unspecified then the numbers will take on the units of the property when they are used (i.e., default value of hours).

Properties: InsideCheckInTime (Expression Property)	
Value	
Default Value	0.0
Candidate References	False
Unit Type	Time
Default Units	Minutes
Specify Unit Type Time (Minutes)	
Appearance	
Display Name	InsideCheckInTime
> Operational Planning	
General	
Name	InsideCheckInTime
Description	
Required Value	True

Figure 4.12: Inside Check-In Properties

**Step 9:** Now, we must specify the processing times at the check-in stations to depend on the particular entity's passenger type, as shown in Figure 4.13. The passenger type is used as the "row reference" in the table. Thus, the specifications needed for the new processing times use the **EStaPassengerType** state variable as:

- At **SrvCurbSideCheckIn**:<sup>32</sup>  
`TablePassenger[ModelEntity.EStaPassengerType].CurbSideCheckInTime`
- At **SrvCheckIn**:  
`TablePassenger[ModelEntity.EStaPassengerType].InsideCheckInTime`

Notice that the content of the `[]`<sup>33</sup> is used to designate the specific row in the table, similar to arrays in programming languages. Unfortunately, the expression editor does not know about the **DATA TABLE** properties since they are user-defined, so you must type the property into the expression. Also, be sure that the processing time units in the check-in stations are specified in minutes.

Properties: SrvCurbSideCheckIn (Server)	
Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Process Type	Specific Time
Processing Time	<b>TablePassenger[ModelEntity.EStaPassengerType].CurbsideCheckInTime</b>
Off Shift Rule	Suspend Processing
> Other Processing Options	

Figure 4.13: Utilizing the Table to Specify the Processing Times

<sup>32</sup> To enter this specification to employ the expression editor during entry is to first put in the table name and property as `TablePassenger.CurbSideCheckInTime` and then insert `[ModelEntity.EStaPassengerType]`. In later chapters, we will not use this approach but directly assign a row in the table directly to the entity.

<sup>33</sup> Note: table rows start at one (i.e., the first passenger type check in time is `TablePassenger[1].InsideCheckInTime`)

**Step 10:** Run the model. Remember, no arrival occurs before 5 am, and the simulation is started at 5 am, so no animation appears before then.

**Question 7:** What are the average wait times at each check-in location (Curbside and Inside)?

---

**Question 8:** What is the average check-in time for a passenger (Curbside and Inside)?

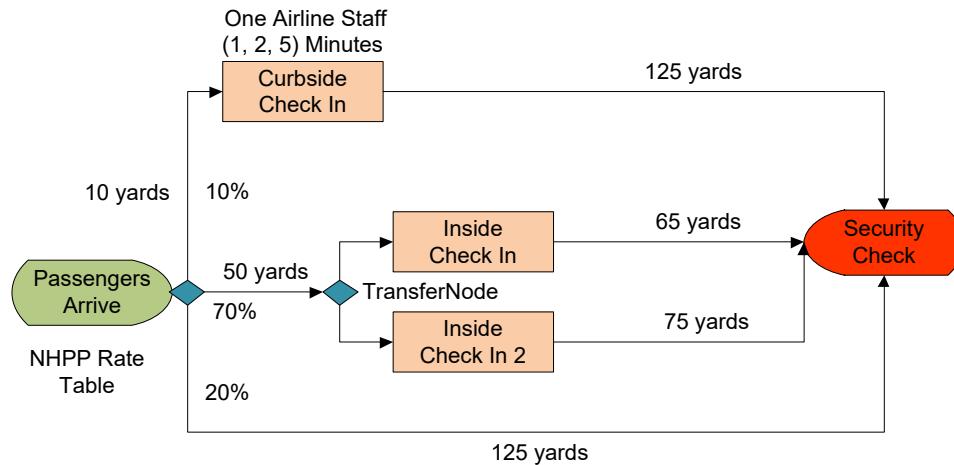
---

**Properties and State Variables:** Before leaving this section, let's reflect on the characteristics of "properties" and "state variables" because they will be used frequently. Properties and state variables can be characteristics of the MODEL or the MODELENTITIES. A "property" of an object is a part of the object's definition and is assigned at the time an object is created, but it can't be changed during the simulation. Properties can be expressions, so their evaluation can yield different values. In contrast, a "state variable" is a characteristic of an object that can be changed during the simulation. Generally, when properties and state variables are associated with the model, they are global in the sense their values can be obtained almost anytime. When properties and state variables are associated with entities, you need access to that model entity to access the value.

SIMIO extensively uses properties and state variables, which you can see by looking at their "definitions" and expanding the (Inherited) Properties/States. Some properties and states are related. For example, the "*InitialPriority*" property of a MODELENTITY becomes the initial value of the "*Priority*" state variable. Properties can be used to give other states their initial values.

#### Part 4.4: More on Branching

At times, the waiting lines at the inside Check-in station seem overcrowded. We decided to add a second Inside Check-in station that passengers can use during rush hours. Thus, passengers will be routed to whichever check-in station has the smallest number in the waiting queue. The distance from this second Check-in station to the Security Check is 75 yards, as seen in Figure 4.14.



**Figure 4.14: Airline Check-in Process with Two Inside Check-In Stations**

**Step 1:** Add the second check-in station by copying and pasting the current **InsideCheckIn**. Call it **SrvCheckIn2**. Assume it has a capacity of one, and its processing times are the same as those for the regular check-in station. Add the path of 75 yards from this second station to **SnkSecurity**.

**Step 2:** We need to model the choice made by the “inside” check-in passengers to go to the check-in station with the smallest number in the queue. There are several ways to model this situation. Perhaps the most obvious or most straightforward way is to add a TRANSERNODE that branches passengers to the shortest queue.

- Delete the original path to the **SrvCheckIn** object from the **SrcPassengersArrive**.
- Insert a TRANSERNODE near both the check-in servers and a path from the **SrcPassengersArrive** to the TransferNode, ensuring the distance from the **SrcPassengersArrive** to this TRANSERNODE is 50 yards. Also, remember that the selection weight is 70 for this path.
- Next, connect the TRANSERNODE to the two check-in servers’ input nodes using CONNECTORS, which will take up zero time and space.
- The TRANSERNODE should specify the *Outbound Link Rule* as “**By Link Weight**.” If you leave it as the “**Shortest Path**,” it will use the selection weights since we have not specified a specific node for the entity to transport to.
- Set the *Selection Weight* properties of the two paths from the TRANSERNODE (the “value” of a “true” expression is one and zero if it is “false”)

(1) TO **SrvCheckIn**:

```
SrvCheckIn.InputBuffer.Contents.NumberWaiting <=
SrvCheckIn2.InputBuffer.Contents.NumberWaiting
```

(2) TO **SrvCheckIn2**:

```
SrvCheckIn2.InputBuffer.Contents.NumberWaiting <
SrvCheckIn.InputBuffer.Contents.NumberWaiting
```

*Question 9:* Which check-in station is chosen if there are “ties” in the size of the InputBuffer (For example, suppose both are empty)?

---

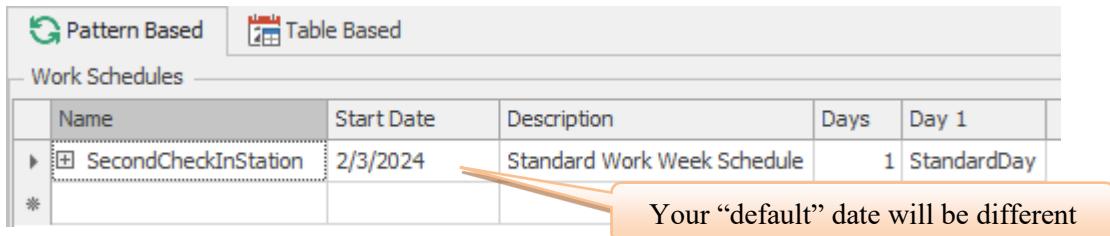
*Question 10:* What is the utilization of **SrvCheckIn2**, and does it make sense to have both check-ins?

---

## Part 4.5: Work Schedules

Suppose you decide only to staff the second inside check-in station from 10 am to 6 pm. This kind of concern requires a work schedule that alters the capacity of a server over time. Work schedules will be built from the “Schedules” component under the “Data” tab.

**Step 1:** Click on the “Work Schedule” button under the “Schedules” icon in the “Data” tab to add a new work schedule. The StandardWeek schedule has already been defined by default, as seen in Figure 4.15. Rename this schedule **SecondCheckInStation**. Change the Days to “1” – this schedule will repeat daily.<sup>34</sup> For each day of the work schedule, you need to specify the work pattern for that day. Therefore, you can have different patterns for different days (i.e., weekends, etc.). The **StandardDay** pattern was specified for *Day 1*. The *Start Date* property can be left as the default since we have a daily pattern.



**Figure 4.15: Creating SecondCheckInStation WORKSCHEDULE**

**Step 2:** In the “Day Patterns” section, you can add DAY PATTERNS by clicking the *Day Pattern* button in the “Create” section. SIMIO defines the **StandardDay** pattern as 8 am to 5 pm with an hour lunch at 12 pm. We only need one work period in the pattern since we start the day at 10 am and end at 6 pm. Select and delete the second work period (1 pm to 5 pm). Then specify the Start Time to be 10 am, the Duration to be eight hours, or the “End Time” to be 6 pm, as seen in Figure 4.16. Specify a “1” for the *Value*, which will specify the capacity of the second inside check-in to be one. If a work period has not been defined for a period in the day pattern, it is assumed to be off shift (i.e., zero capacity).

---

<sup>34</sup> If there are seven days in the pattern, then the days of the week will appear. In our case there is only the Day 1 pattern. SIMIO will repeat the pattern based on the number of days in the WORK SCHEDULE. For each day in the work schedule you need to specify a pattern. Also, we have the ability to specify exceptions to the day pattern or work periods for particular dates.

Day Patterns						
Name	Description					
StandardDay	Standard 8-5 Work Day					
Work Periods						
	Start Time	Duration	End Time	Value	Cost Multiplier	Description
*	10:00 AM	8 hours	6:00 PM	1	1	

Figure 4.16: Setting a Day Pattern with Different Work Periods

**Step 3:** Go to the “Facility” view and click on the **SrvCheckIn2** object. Change the *Capacity Type* to “**WorkSchedule**” and the *Work Schedule* property to **SecondCheckInStation** as in Figure 4.17.

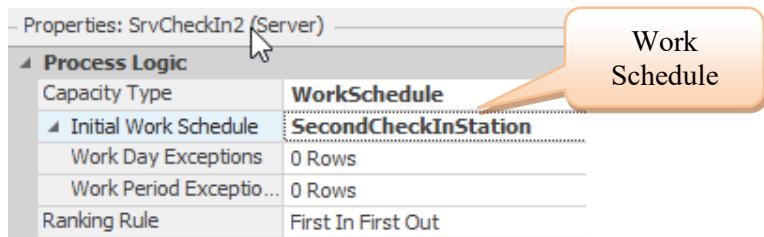


Figure 4.17: Specifying a Work Schedule

**Step 4:** Now, we need to be sure that people who choose the inside check-in only go to the **SrvCheckIn2** station if it is open (i.e., only between hours 10 and 18). This requires another **TRANSFERNODE**, as seen in Figure 4.18. Connect the original **TRANSFERNODE** to the new one via a Connector and connect the new **TRANSFERNODE** to the inputs of both check-in stations.

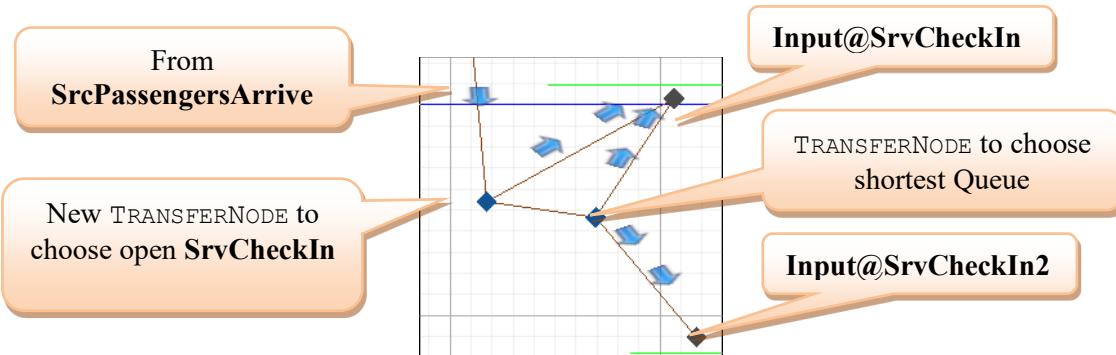


Figure 4.18: Adding Another TransferNode to Handle the Logic

- The new **TRANSFERNODE** should determine routing *By Link Weight* as well.
- Path to **Input@SrvCheckIn** has zero length<sup>35</sup> and *Selection Weight* property: `Run.TimeNow <=5 || Run.TimeNow>=13`<sup>36</sup>

<sup>35</sup> May be easier to use a CONNECTOR instead of a PATH – CONNECTORS have zero length and take zero time to traverse.

<sup>36</sup> Note, SIMIO utilizes `||` as the “Or” logical operator and `&&` as the “And” logical operator. Also, this expression works only here since we are simulating just one day and 10 A.M. represents five hours into the simulation since we are starting at 5 A.M. If we were simulating more than one day, then we would need to specify `Math.Remainder(Run.TimeNow, 24) * 24` which converts the running time into a number between 0 and 24.

- Path to TRANSERNODE to choose the shortest queue has zero length and *Selection Weight* property of Run.TimeNow > 5 && Run.TimeNow < 13<sup>37</sup>

**Step 5:** Note that the default time units are hours when there is no context. Now, run the model and obtain these basic statistics.

*Question 11:* What is the average Check-In time for a Passenger?

---

*Question 12:* What is the average wait time for Curbside Check-In?

---

*Question 13:* What are the average wait times for two inside Check-In stations?

---

*Question 14:* What is the utilization for an employee at the Curbside Check-in?

---

*Question 15:* What are the utilizations for an employee at the two onside Check-in stations?

---

**Step 6:** You may have noticed that in the simulation, there are entities in the queue of the **SrvCheckIn2** that are in line when the station closes. These are marooned in this model. We will consider how to handle this kind of problem in a later chapter.

*Question 16:* What would have to change to the model each time the work schedule of the second check-in server was modified?

---

**Step 7:** If we want to determine the best starting and ending times to open up the second check-in station, the link weights using the Run.TimeNow expression would have to be changed to match the new start and end times each time. A better approach would be to use the server's capacity so that as it goes to zero, we need to close the path and open it when the capacity goes to one. Change the link weights according to and rerun the model, comparing the previous results.

- Path to **Input@SrvCheckIn** *Selection Weight* property: SrvCheckIn2.Capacity == 0<sup>38</sup>
- Path to TRANSERNODE to choose the shortest queue has zero length and *Selection Weight* property of SrvCheckIn2.Capacity > 0

---

<sup>37</sup> We could have modified the original link weights and avoided having to add the new paths and transfer node. Specify the expression to be (TimeNow <= 5 || TimeNow >= 13) || SrvCheckIn.InputBuffer.Contents.NumberWaiting <= SrvCheckIn2.InputBuffer.Contents.NumberWaiting for the link weight for the path to **SrvCheckIn** and for the path link weight to **SrvCheckin2** to (Run.TimeNow > 5 && Run.TimeNow < 13) && SrvCheckIn2.InputBuffer.Contents.NumberWaiting < SrvCheckIn.InputBuffer.Contents.NumberWaiting.

<sup>38</sup> Note, SIMIO utilizes a double equal sign (i.e., “==”) as the logical equal.

## Part 4.6: Commentary

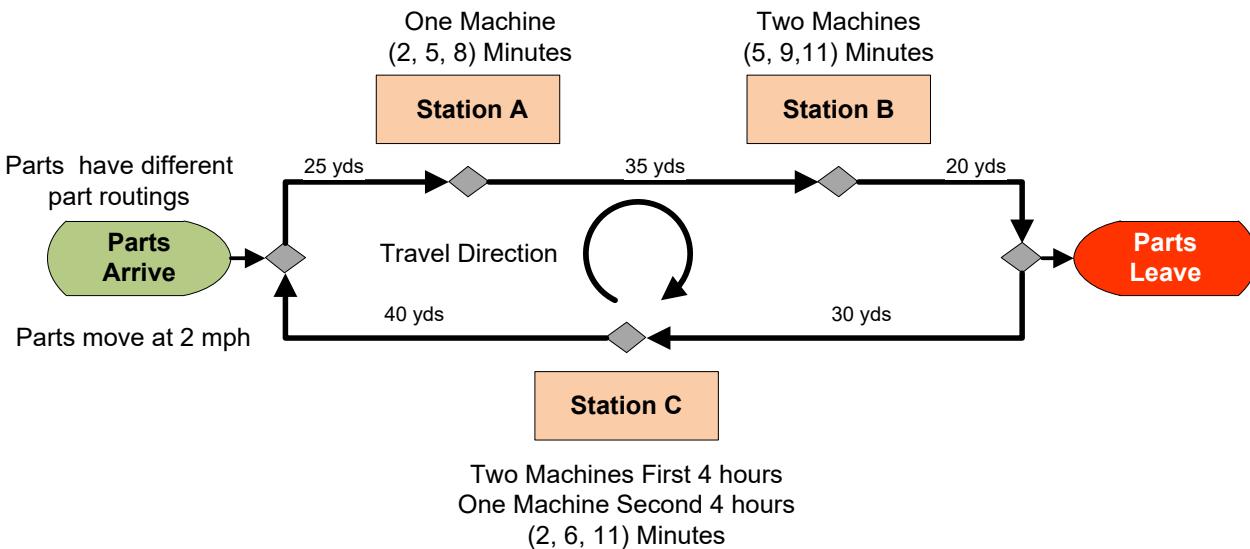
- With the very high utilization and long wait times, one would use the model to determine the appropriate capacity at each inside check-in station to make the waiting time reasonable.
- Specifying the time-varying arrival pattern in SIMIO requires fixed units for the arrival rate, namely arrivals per hour.
- Please pay special attention to the SIMIO specification of cumulative probability and its value for the Random. Discrete and Random.Continuous distributions.
- While work schedules in SIMIO offer considerable flexibility in specifying exceptions, etc., they are somewhat complicated to input. One has to be careful when specifying the correct times and duration. Also, be cautious when specifying the time to begin and end the simulation run.
- In SIMIO, when a server's capacity goes to zero while there is an entity (or entities) being processed, SIMIO chooses, by default, to "Ignore" this so that the items "in process" will be processed while at the same time, the server capacity goes to zero. If another behavior is desired, then you will need to implement it using more advanced concepts, which will be described later.
- Note that if the length of the simulation replication exceeds the work schedule time, the work schedule is repeated until the replication ends.

# Chapter 5

## Data-Based Modeling: Manufacturing Cell

Most simulation models employ substantial amounts of data in their development and execution. In SIMIO, employing a data table, as seen in the previous chapter, provides a convenient means to store and recall the data. In this chapter, we will use tables more extensively by reconstructing a manufacturing cell consisting of several workstations.

A small manufacturing cell comprises three workstations through which four different types of parts are processed. The workstations A, B, and C are arranged in such a way that the parts must follow the one-way circular pattern, as shown in the layout in Figure 5.1. For instance, if a part is finished at Station C, it would have to travel the distance to Station A and then the distance to Station B before it could travel to where it exits the system. We want to simulate a five-day week with eight hours of operation a day for a total of 40 hours.



**Figure 5.1: Workstation Layout**

All parts arrive at the “Parts Arrive” location and leave at the “Parts Leave” location. Travel between workstations is a constant two miles per hour. Note that SIMIO’s default speed for entities is *meters per second*. The distances (yards) between workstations are shown in Table 5.1 and Figure 5.1.

**Table 5.1: Distance between Workstations**

WorkStation Path	Distances (yds.)
Parts Arrive to Station A	25
Station A to Station B	35
Station B to Parts Leave	20
Parts Leave to Station C	30
Station C to Parts Arrive	40

Each part type arrives as follows.<sup>39</sup>

- Part 1's arrive randomly with an average of 15 minutes between arrivals.
- Part 2's have an interarrival time that averages 17 minutes with a standard deviation of 3 minutes.
- Part 3's can arrive with an interarrival time of anywhere from 14 to 18 minutes apart.
- Part 4's arrive in batches of five exactly 1 hour and 10 minutes apart.

*Question 1:* What distributions will you choose to model these four arrival processes?

---

Each part (entity) type is sequenced (routed) across the stations differently. In fact, not all part types are routed through all machines, as seen in the sequence order given in Table 5.2.

**Table 5.2: Part Sequences**

Step	1	2	3
Part 1	Station A	Station C	
Part 2	Station A	Station B	Station C
Part 3	Station A	Station B	
Part 4	Station B	Station C	

***Server Properties:***

- Station A has one machine with a processing time of two to eight minutes and a likely (modal) time of five minutes.
- Station B has two machines, each with a processing time of five to eleven minutes and a likely time of nine minutes.
- Station C has two machines operating during the first four hours of each day and one operating during the last four hours of the day. Each has a processing time ranging between two and eleven minutes and a mode of six minutes.

*Question 2:* What distributions will you choose to model the three-station processing times?

---

## Part 5.1: Constructing the Model

We will build the SIMIO model using four SOURCES, three SERVERS, and several types of links.

**Step 1:** Insert four SOURCES named **SrcParts1**, **SrcParts2**, **SrcParts3**, and **SrcPart4** to create the four different part types (i.e., drag four MODELENTITIES onto the model canvas named **EntPart1**, **EntPart2**, **EntPart3**, and **EntPart4**).<sup>40</sup> Four SOURCES are necessary because each part has its own arrival process. Make sure to change the *Entity Type* property of the four SOURCES to produce the appropriate entity type.

**Step 2:** Insert three SERVERS (i.e., one for each workstation named **SrvStationA**, **SrvStationB**, and **SrvStationC**) and one SINK named **SnkPartsLeave**. Be sure to name all the objects, including the entities, so it will be easy to recognize them later, as seen in Figure 5.2.

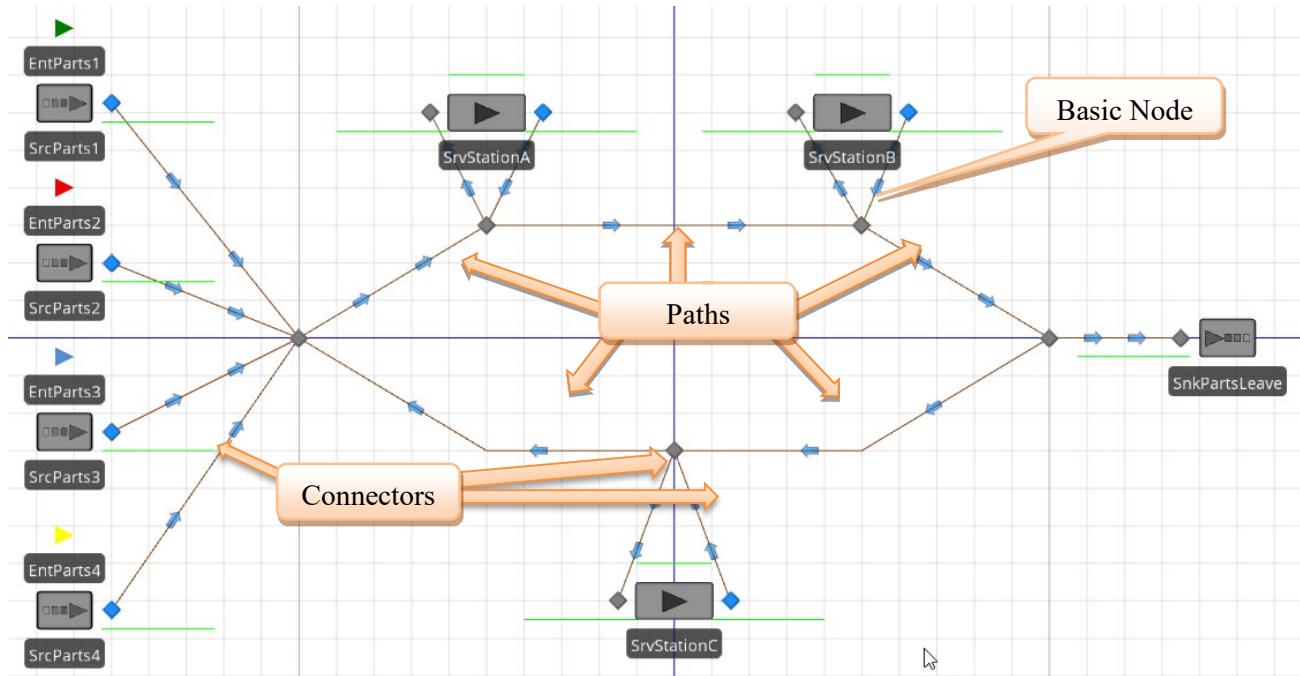
---

<sup>39</sup> For the moment, we will let you think about the distributions that these parameters may represent.

<sup>40</sup> To do this efficiently, drag one SOURCE on to the facility and name it **SrcParts**. Then copy and paste it three times which will take advantage of the SIMIO naming scheme (i.e., it will create **SrcParts1**, **SrcParts2**, and **SrcParts3**). Now, rename the first source **SrcParts4**. Repeat this method for the four entities starting with **EntPart**.

**Step 3:** The best method to simulate the circular travel pattern would probably be to use a **BASICNODE** at each station's entry and exit point.<sup>41</sup> This allows the entities to follow the sequence pattern without going through an unneeded process. A **BASICNODE** is a simple node to connect two different links, while a **TRANSFERNODE** can connect different links as well but provides the ability to select a destination, path, and/or require a **TRANSPORTER**.

**Step 4:** Now connect all the objects using **CONNECTORS** and **PATHS**. Note that we will use connectors because there is no time lapse or distance between the circular track and the server objects. The distances between stations will be calculated using the five paths connecting each entry and exit node. Be sure that you draw the connectors and paths in the right direction – zoom in on your model to see the “arrows” as shown in Figure 5.2 (i.e., links are entering the input nodes of the **SERVERS** while leaving the output nodes).



**Figure 5.2: Initial Model**

**Step 5:** Change the properties of each entity so that they now travel at the desired speed of two miles per hour.<sup>42</sup> Also, change the color of the entities so you can distinguish between the four-part types by choosing the color on the “Drawing” tab and then selecting the specific entity.

**Step 6:** Change the properties of the four source objects so that they correspond to the correct parts and have the desired interarrival times and entities per arrival. We chose Exponential, Normal, and Uniform as the distributions of interarrival times for part types one, two, and three, respectively, with the parameters given. For part type four, we chose a constant interarrival time of 70 minutes and set the *Entities Per Arrival* property to 5, as seen in Figure 5.3.

---

<sup>41</sup> A **TRANSFERNODE** used to enter/exit from a station can disrupt the SIMIO internal sequencing of sequence steps if one is not careful.

<sup>42</sup> Select all four entities using the *Ctrl* button and specify the speed once for all entities.

Properties: SrcParts4 (Source)	
Entity Arrival Logic	
Entity Type	EntParts4
Arrival Mode	Interarrival Time
► Time Offset	0.0
▲ Interarrival Time	70
Units	Minutes
Entities Per Arrival	5

Figure 5.3: Specifying Parameters for the SrcPart4

**Step 7:** Set the distances for each of the five paths to correspond to the distances between the stations. You will have to set the *Drawn To Scale* property for each to “*False*” to do this and then specify the logical length and units of type “Yards.”

**Step 8:** Set the processing times for all the stations using the Pert (BetaPert) distribution. In general, we prefer the Pert distribution to the Triangular distribution. It is specified like the Triangular (minimum, most likely, maximum) but has “thinner” tails, which, we believe, better represents the expected behavior of the processing time distribution. See Appendix A for more information.

## Part 5.2: Setting Capacities

We will need to set the capacities for the three workstations, A, B, and C.

**Step 1:** Set the capacities for A and B to one and two, respectively. For Station C, we will need to add a work schedule to accommodate the changes in capacity.

**Step 2:** Go to the “*Data*” tab, select “*Schedules*,” and modify the default schedule (**StandardWeek**) and the default day pattern (**StandardDay**). Rename the work schedule to **ScheduleForC**<sup>43</sup> and leave the *Start Date* property alone to remain the default date on which the simulation will start in the *Run* setup.

We aim to simulate a five-day week with eight hours of operation a day for a total of 40 hours. Because we do not want to include the idle time (i.e., 12 am to 8 am, 12 pm to 1 pm, and 5 pm to 12 am) in the statistics, we will simulate 40 straight hours, which is equivalent. SIMIO will repeat the work cycle over how many days are in the work pattern. Because our simulation will start at 12 am and run for 40 hours, we want to set the Days in *Work Schedule* property to 1 and let the cycle repeat after the first 24 hours.

Work Schedules					
	Name	Start Date	Description	Days	Day 1
►	ScheduleC	2/3/2024	Standard Work Week Schedule	1	StandardDay
*					

Figure 5.4: Setting up the WorkSchedule

Now select the “*Day Patterns*” tab to modify the **StandardDay** day pattern to match the one in Figure 5.5. Change the *Start Time* property of the first row to 12:00 AM, which will cause the *Duration* to change to match the *End Time*. Change it back to “4 hours” and click the *Enter*. After changing the capacity value to two, repeat this process for the second row using one as the capacity value. Next, add four new work periods by specifying the *Start Time*, the *Duration*, and the capacity value.

<sup>43</sup> Double clicking the *Standard Week* name will allow you to change the name.

Day Patterns										
Name	Description									
► StandardDay Standard 8-5 Work Day										
Work Periods										
Start Time	Duration	End Time	Value	Cost Multiplier						
12:00 AM	4 hours	4:00 AM	2	1						
4:00 AM	4 hours	8:00 AM	1	1						
8:00 AM	4 hours	12:00 PM	2	1						
12:00 PM	4 hours	4:00 PM	1	1						
4:00 PM	4 hours	8:00 PM	2	1						
8:00 PM	4 hours	12:00 AM	1	1						
*										

**Figure 5.5: Modifying the Day Pattern to Represent Three Eight-Hour Days**

**Step 3:** Go back to the “Facility” tab and set the *Capacity Type* property of Station C to “WorkSchedule” using your work schedule, namely **ScheduleForC**.

Properties: SrvStationA1 (Server)	
▲ Process Logic	
Capacity Type	WorkSchedule
► Initial Work Schedule	ScheduleC

**Figure 5.6: Specifying the WorkSchedule for the StationC**

*Question 3:* How many hours during a 24-hour day is the capacity of Station C equal to two?

**Step 4:** Save and run the model for 40 hours, observing the different entities' paths.

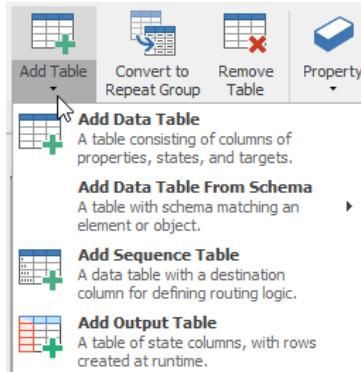
*Question 4:* How long is each part type in the system (enter to leave)?

*Question 5:* Why do you think the time in the system is large for all four-part types?

### Part 5.3: Incorporating Sequences

The way the current model is set up, entities will branch at nodes based on link weights and the direction of the link. However, we want the entities to follow a desired sequence of stations as seen in Table 5.2. We want to be sure that entities will continue along the circular pattern until they reach the node that takes them to their particular station. To accomplish this, we will use “Sequence Tables” (from the “Data” tab).

**Step 1:** Click the “Add Sequence Table” button under the drop-down from “Add Table” in the “Data” tab, as seen in Figure 5.7, and rename the new table **SequencePart1** to correspond to the first entity type.



**Figure 5.7: Adding a Sequence Table**

**Step 2:** Now add the sequence of nodes that you want Part 1 to follow (i.e., Station A, Station C, and Exit). You only need to include the visited nodes at each station and the exit. Once the entity's sequence has been set, it will always travel the shortest route to get to the first node on the list. The correct sequence for **Part1** is given in Figure 5.8.<sup>44</sup>

Sequence Part1	
	Sequence
1	Input@SrvStationA
2	Input@SrvStationC
3	Input@SnkPartsLeave
▶	

**Figure 5.8: Sequence for Part1**

**Step 3:** Next, add three more sequence tables to correspond to the remaining three entities (i.e., **SequencePart2**, **SequencePart3**, and **SequencePart4**). Set the sequences for each of the other three parts in their corresponding sequence tables as well. Make sure the last node is associated with the **SINK**. Note that the stations' names may appear differently in your sequence if you name them differently.

**Step 4:** Return to the “Facility” tab and click on the **EntPart1** entity. Set the *Initial Sequence* property to the “**SequencePart1**” under the “*Routing Logic*” section, which you just created for “Part 1,” as seen in Figure 5.8. Please do the same for each of the other entity types and their associated sequences.

Routing Logic	
Initial Priority	1.0
Initial Sequence	<b>SequencePart1</b>

**Figure 5.9: Specifying an Entity to follow a Particular Sequence**

**Step 5:** Do the same for each entity type and their associated sequence. A better method is to use the “Properties Spreadsheet,” as seen in Figure 5.10. Right-click one of the MODELENTITIES and select the “*Object Property Spreadsheet*” button. Next, select the “*Routing Logic*” tab and specify each entity's sequence table.

<sup>44</sup> Note the input nodes of the associated **SERVERS** and **SINK** are specified rather than the actual object. Note the names of the nodes are in the form **Input@ObjectName**.

Properties Spreadsheet - 4 ModelEntity Objects				
	Travel Logic	Routing Logic	Financials	Population
	Instance Name	Initial Priority	Initial Sequence	
▶	EntParts1	1.0	SequencePart1	
	EntParts2	1.0	SequencePart2	
	EntParts3	1.0	SequencePart3	
	EntParts4	1.0	SequencePart4	

**Figure 5.10: Using the Properties Spreadsheet to Set Properties of All Four Entities**

**Step 6:** For each of the seven TRANSERNODES (i.e., the output nodes at each source and server), change the *Entity Destination Type* property from “Continue” to “By Sequence” for each one.<sup>45</sup> The first time an entity passes through a transfer node where its destination is “By Sequence,” the node sets the destination of the entity to the first row index of the SEQUENCE TABLE. After that, every time an entity passes through a transfer node with the “By Sequence” destination type, the row index of the table is increased by one, and the destination of the entity is set to the next node in the table.<sup>46</sup>

**Table 5.3: Definition of Basic and TransferNodes**

 <b>BasicNode</b>	Simple node to support the connection between links. Often used for Input nodes of objects (e.g., SINK, SERVER, etc.).
 <b>TransferNode</b>	Supports connection between links but has the ability to select entity destination as well as request a transporter. Output nodes of objects (SOURCE, SERVER, etc.) are TRANSERNODES.

*Question 6:* Will passage through a BASICNODE also change the row index of the SEQUENCE TABLE?

---

*Question 7:* Will passage through a TRANSERNODE always change the row index of the SEQUENCE TABLE?

---

**Step 7:** It will also help ensure they follow the correct sequence. Run the simulation for 40 hours at a slow speed to ensure each part follows its sequence appropriately.

*Question 8:* Turn off the arrivals of all but one part type by changing the *Entities Per Arrival* property of the SOURCE object to zero for all but one SOURCE. Does it follow the expected sequence?

---

**Step 8:** If you turned off any source, turn them back on by changing the *Entities Per Arrival* property back to the original value. Save the current model and run it for 40 hours.

*Question 9:* How long is each part type in the system now (enter to leave)?

---



---

<sup>45</sup> To set them all at once, press the *Ctrl* key while selecting all seven TRANSERNODES and then set the *Entity Destination* to “By Sequence.”

<sup>46</sup> If the current row index is on the last row and the entity enters a TRANSERNODE that has “By Sequence,” the row index will start over at one. Also, if an entity enters a TRANSERNODE that has “By Sequence,” before reaching the current destination node, the row index will move to the next row in the sequence table.

## Part 5.4: Embellishment: New Arrival Pattern and Processing Times

We now learn that the parts arrive in one stream of arrivals (i.e., with a mean arrival rate of 6 per hour, Poisson distributed<sup>47</sup>) instead of four. There are still four-part types, but the type is random based on the percentages specified in Table 5.4. The number of each that arrives doesn't change (i.e., meaning Part 4 arrives five at a time).

**Table 5.4: Part Type Percentage**

Part Type	Percentages	Number to Arrive
Part 1	25%	1
Part 2	35%	1
Part 3	25%	1
Part 4	15%	5

*Question 10:* When do you think using a single source to model entity arrivals is preferred over multiple sources?

---

Previously, the processing times at the various stations were the same regardless of the part type. Now, we discover that the processing times depend on the particular part type, as shown in Table 5.5. In this table, the sequences are the same as before, but the processing times have been added here.

**Table 5.5: Entity Sequences and Processing Times (in Minutes)**

Step	1	2	3
Part 1	Station A (Pert (2,5,8))	Station C (Pert(2,6,11))	
Part 2	Station A (Pert (1,3,4))	Station B (Uniform (5 to 11))	Station C (Uniform (2 to 11))
Part 3	Station A (Triangular (2,5,8))	Station B (Triangular (5,9,11))	
Part 4	Station B (Pert(5,9,11))	Station C (Triangular (2,6,11))	

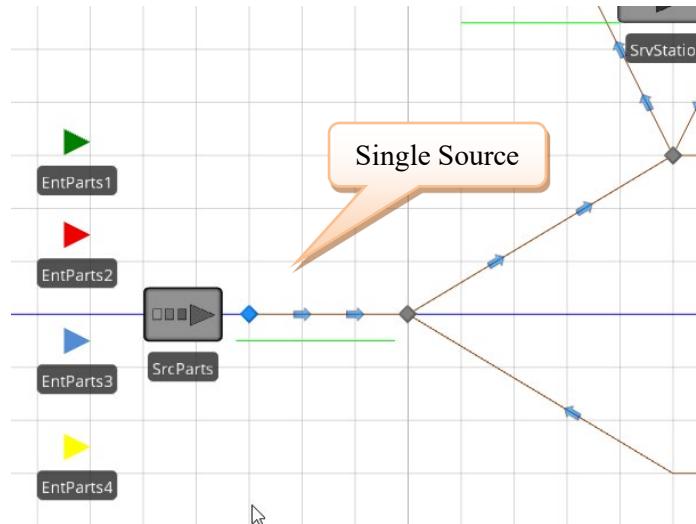
*Question 11:* Do you think that it is realistic that the processing time for Part 1 on Station A might be different for Part 2 on that same Station A?

---

**Step 9:** Now delete all but one of the SOURCES referring to Figure 5.11 as an example and rename the one SOURCE as **SrcParts**. We will return to specifying this source after we specify the processing times.

---

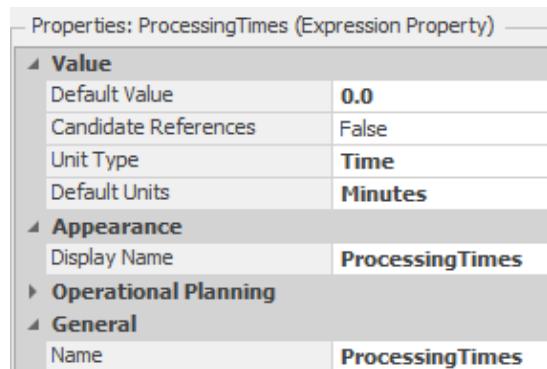
<sup>47</sup> Remember that this Poisson arrival rate is equivalent to an Exponential interarrival time with a mean of 10 minutes. SIMIO wants the interarrival time, not the arrival rate – so be prepared to make the conversion.



**Figure 5.11: Manufacturing Cell Model**

**Step 10:** The processing times will be added to the original sequencing tables to model the processing times, which are dependent on the part type. When the entity is assigned a destination (i.e., a row in the sequence table is selected), the processing time column can be accessed for that row. (Note: you could use a more complicated expression for the processing time by checking to see if this is a Part A, then make the processing time X, etc., but it is not very easy if there is a large number of parts).

**Step 11:** Go to the “Data” tab and click on the **SequencePart1** table, which is associated with Part 1 types. Since the processing times can be any distribution (i.e., any valid expression), add a new *Expression* type column from the *Standard Property* drop-down named **ProcessingTimes**. Ensure the column has the appropriate units by making the *Unit Type* “Time” and the *Default Units* “Minutes.”



**Figure 5.12: Adding an Expression Column to the Sequence Tables**

**Step 12:** Set the Processing times for **SrvStationA** (i.e., the first row) to a `Random.Pert(2, 5, 8)`, **SrvStationC** (i.e., the second row) to a `Random.Pert(2, 6, 11)` and finally, the **SnkPartsLeave** or third row will remain zero, as seen in Figure 5.13.

Sequence Part1		Sequence Part2	Sequence Part3	Sequence Part4
	Sequence	ProcessingTimes (Minutes)		
► 1	Input@SrvStationA	Random.Pert(2,5,8)		
2	Input@SrvStationC	Random.Pert(2,6,11)		
3	Input@SnkPartsLeave	0.0		
*				

Figure 5.13: New Sequence Table for Part Type 1

**Step 13:** Repeat the last two steps for the remaining three-part sequence tables using the appropriate processing times from Table 5.5.

**Step 14:** In order to have a single source produce multiple types of entities, a new data table has to be created to specify the part type entities and the part mix percentages. Add a new data table called **TableParts** with four columns with the specified types, as seen in Table 5.6.

Table 5.6: Column Definitions

Column	Column Name	Column Type
1	PartType	Object Reference Property → Entity
2	ProcessingTimes	Property → Expression
3	PartMix	Property → Integer (or Real)
4	NumberToArrive	Property → Integer (or Real)

**Step 15:** Next, specify each entity as a new row with the appropriate processing times, part mix percentages, and sequence table as shown in Table 5.7.

Table 5.7: Content of TableParts

Table Parts		Sequence Part1	Sequence Part2	Sequence Part3	Sequence Part4
	Part Type	ProcessingTimes (Minutes)	Part Mix	Number to Arrive	
1	EntParts1	SequencePart1.ProcessingTimes	25	1	
2	EntParts2	SequencePart2.ProcessingTimes	35	1	
3	EntParts3	SequencePart3.ProcessingTimes	25	1	
► 4	EntParts4	SequencePart4.ProcessingTimes	15	5	

Specifying `SequencePart1.ProcessingTimes` states that the processing time will come from the column associated with `ProcessingTimes` in the table `SequencePart1` for the current row.

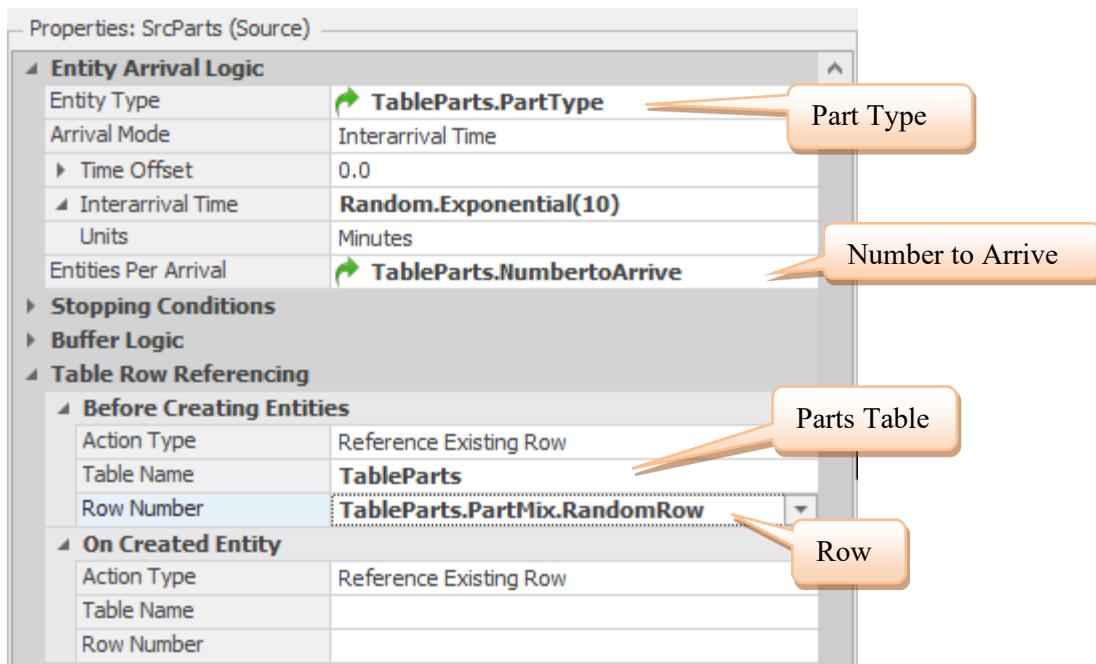
**Step 16:** In order to use the dependent processing times, change each of the processing time expressions in the three stations to `TableParts.ProcessingTime`. This logic specifies that the processing time will come from the `ProcessingTimes` column of the data table `TableParts`. As the parts move through their individual sequence table (i.e., row to row), it will retrieve the processing time for the current row.

The property is now a reference property type `TableParts.ProcessingTime` which can only be changed back to a regular property by right-clicking on the label (i.e., `Processing Time`) and resetting it.

**Step 17:** The *Entity Type* must be changed from a specific entity type to generate multiple part types from the same source. In the `SOURCE`, specify the *Entity Type* to come from the `TableParts.PartType`. This will allow the *Entity Type* to come from one of the four entities defined in the table.

**Step 18:** Set the interarrival time of entities to be an Exponential with a mean of ten minutes (i.e., same as a Poisson arrival rate of six per hour). Set the *Entities Per Arrival* to **TableParts.NumberToArrive**.

**Step 19:** The part mix percentages will be utilized to determine which specific part type is created. The SOURCE has a property category called “Table Reference Assignment,” which can assign a table and a row of that table to the entity.<sup>48</sup> There are two possible assignments: *Before Creating Entities* and *On Created Entity*. We want to do the assignment before the entity is actually created so that the SOURCE can create the correct entity type and number to arrive. Therefore, specify the table shown in Figure 5.14.



**Figure 5.14: Making the Table Reference Assignment**

You specify which table (e.g., **TableParts**) and which row in the table to associate with the object. In this case, the row is determined randomly by using the **PartMix** column and specifying that a random row should be generated (**TableParts.PartMix.RandomRow**).<sup>49</sup>

**Step 20:** Save and run the simulation model for 40 hours at a slow speed to ensure each part follows its sequence appropriately and answer the following questions.

*Question 12:* How long is each part type in the system (enter to leave)?

---

*Question 13:* What is the utilization of each of the SERVERS?

---

*Question 14:* What is the overall time for all part types in the system (utilize the **SINK** statistics)?

---



---

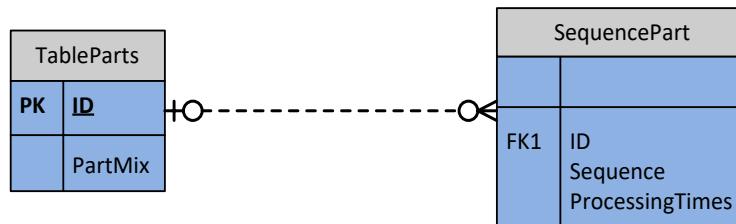
<sup>48</sup> In later chapters we will utilize *Processes* and the *Set Row* step to assign a table to an entity.

<sup>49</sup> SIMIO will normalize the numbers so the probability sums to one. Therefore, you may specify whole numbers 25, 25, and 50 rather than percentages 0.25, 0.25, and 0.5.

## Part 5.5: Using Relational Tables

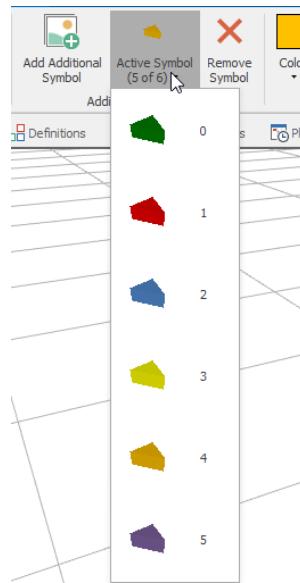
The original model utilized just four-part types; however, we may need to add additional parts. This would require adding a new entity and rows into the TableParts and creating a new sequence table for each new part. Each addition now requires design time to continue updating the model and adding sequence tables. Instead, can we drive these specifications from a spreadsheet containing the number of parts and their sequences? Or is there an easier way to implement the creation and sequencing of parts?

One inventive modeling construct in SIMIO is using *relational data tables*. SIMIO has implemented limited database capabilities that can facilitate the modeling of relationships. Figure 5.15 shows a “relational diagram”<sup>50</sup> where the **TableParts** data table is related to the **SequencePart** Table via the **ID** column. **ID** is a *primary key* (PK) in the **TableParts**, which uniquely identifies each row in the table (i.e., there cannot be any duplicates). The child table (i.e., **SequencePart**) inherits the primary key from the parent table (i.e., **TableParts**), which is denoted as a *foreign key* (FK). A particular part can have many rows (records) in the child table.



**Figure 5.15: Setting up a Relation between Table Parts and the Sequence Part**

**Step 1:** First, we will no longer need just four-part types since we want to make this a more generic model. Delete all but one of the part-type ENTITIES and rename the remaining entity EntParts. Add five additional symbols from the *Additional Symbols* section, making sure to color each new entity as seen in the rotated Figure 5.16.<sup>51</sup>



**Figure 5.16: Adding Additional Symbols for the Part Entity**

<sup>50</sup> More formally it is called an Entity Relationship Diagram, but the word “Entity” is used differently than it is used in SIMIO.

<sup>51</sup> Note the symbols start with “0” index.

**Step 2:** For the **SrcParts**, set the *Entity Type* property to **EntParts**.<sup>52</sup>

**Step 3:** From the *Data* tab, delete all the columns of the **TableParts** except for the *Part Mix* and *Number to Arrive* columns. Insert a new *Integer Standard Property* named **ID**. From the *Edit Column* section, you can move the new column to the left and set the column as a PK using the *Set Column As Key* button in the same section, as seen in Figure 5.17. Set the **ID** for each part type to a unique number (i.e., 1, 2, 3, and 4).

Table Parts		Sequence Part	
	ID	Part Mix	Number to Arrive
► 1	1	25	1
2	2	35	1
3	3	25	1
4	4	15	5
*			

Figure 5.17: Parent Table **TableParts** with the **ID** Primary Key

**Step 4:** Select the **SequencePart1** sequence table and rename it **SequencePart**. From the *Columns* section, select the *Foreign Key Property* button. Set the properties for the column to specify **TableParts.ID** is the primary table key to relate the column, too, as seen in Figure 5.18. Change the name of the column to **ID**.

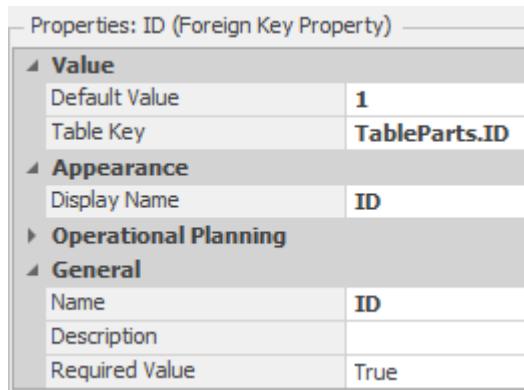


Figure 5.18: Specifying the Foreign Key Property.

**Step 5:** Set the **ID** for all the current rows to one.

**Step 6:** Select the **SequencePart2** tab and then highlight all the rows by selecting the first row using the row selector and dragging it down. Then, copy all the rows and transverse them back to the **SequencePart** table. Select the last row and then paste in all the values. Change the IDs of this new part to two.

**Step 7:** Repeat for the last two-part sequence tabs, setting their IDs to three and four, respectively, as seen in Figure 5.19.

---

<sup>52</sup> Right click the property and reset it back from the reference property.

	Sequence	ProcessingTimes (Minutes)	ID
1	Input@SrvStationA	Random.Pert(2,5,8)	🔑 1
2	Input@SrvStationC	Random.Pert(2,6,11)	🔑 1
3	Input@SnkPartsLeave	0.0	🔑 1
4	Input@SrvStationA	Random.Pert(1,3,4)	🔑 2
5	Input@SrvStationB	Random.Uniform(5,11)	🔑 2
6	Input@SrvStationC	Random.Uniform(2,11)	🔑 2
7	Input@SnkPartsLeave	0.0	🔑 2
8	Input@SrvStationA	Random.Triangular(2,5,8)	🔑 3
9	Input@SrvStationB	Random.Triangular(5,9,11)	🔑 3
10	Input@SnkPartsLeave	0.0	🔑 3
11	Input@SrvStationB	Random.Pert(5,9,11)	🔑 4
12	Input@SrvStationC	Random.Triangular(6,9,11)	🔑 4
13	Input@SnkPartsLeave	0.0	🔑 4
*			

Figure 5.19: The New Sequence Parts Table

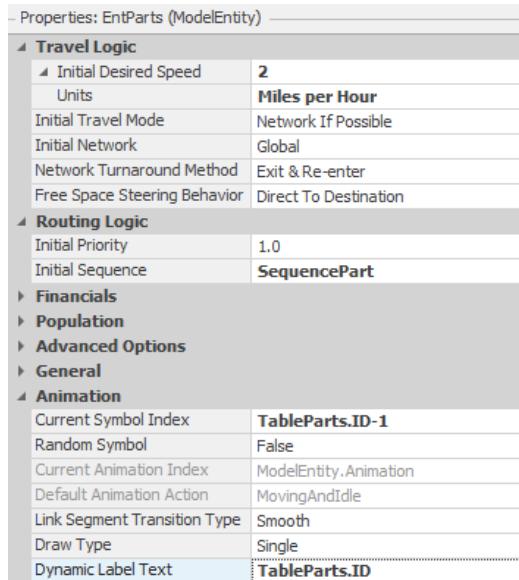
**Step 8:** Once the table is set up, select the **TableParts** table again and observe that each row now has an expander (⊕) associated with it. Figure 5.20 shows the related records for “Part type 0.” If an entity is assigned a row from the **TableParts** table, the related records in the **SequencePart** table will automatically be assigned.

	ID	Part Mix	Number to Arrive	
1	1	25	1	
Sequence Part				
1	Input@SrvStationA	Random.Pert(2,5,8)	🔑 .	
2	Input@SrvStationC	Random.Pert(2,6,11)	🔑 .	
3	Input@SnkPartsLeave	0.0	🔑 .	
*			🔑 .	
2	2	35	1	
3	3	25	1	
4	4	15	5	

Figure 5.20: Seeing the Related Table Information

**Step 9:** We must modify the entity properties to utilize the new sequence table and **ID** to change the picture symbol. Under the *Animation* section, change the Current Symbol Index to “TableParts.ID-1” so the

color will change appropriately and the Dynamic Label Text to “TableParts.ID”.<sup>53</sup> Also, ensure the *Initial Sequence Property* is set to “**SequencePart**,” as seen in Figure 5.21. Note that the entity will only receive the related records, not the entire **SequencePart** table.



**Figure 5.21: Modifying the EntParts Property to Utilize the New Tables**

**Step 10:** For each of the three server stations, change the *Processing Time* to the reference property **SequencePart.ProcessingTimes** rather than **TableParts.ProcessingTimes** since this column is now only in the related table.

**Step 11:** Save and execute the model.

*Question 15:* What is the utilization of each of the SERVERS now?

---

*Question 16:* What is the overall time for all part types in the system?

---

*Question 17:* Is there anything different in this model from the one in the previous section?

---

## Part 5.6: Creating Statistics

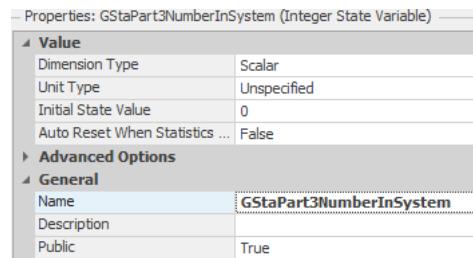
SIMIO automatically produces statistics for various objects within the simulation. For example, statistics are automatically included for the MODELENTITY number and time in the system. For any server, there is a scheduled utilization, a number in the INPUTBUFFER station, the waiting time in the station, etc. However, despite their comprehensive nature, that may be insufficient. For example, our latest model has only one entity type, so the MODELENTITY statistics are not separated by the part type. What if we wanted to know, on average, how many Part 3's are in the system during simulation, and what is the average time these parts spend in the system?

---

<sup>53</sup> If we had a lot of parts, we can utilize just the *Dynamic Label Text* to display the ID as the part moves through the network making this work for any number of parts. This concept will be explored in later chapters.

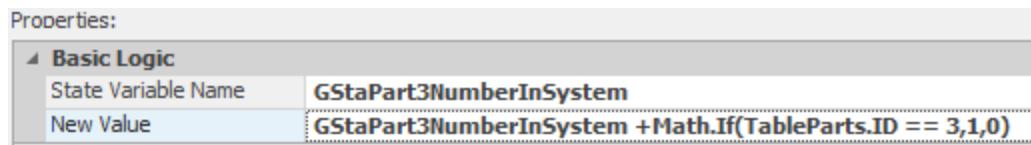
The two major statistical categories are TALLY statistics and STATE statistics. The TALLY statistic is based on observations like waiting time, time in the system, cycle time, etc. The STATE statistic, such as number in queue, number in inventory, etc., is a time-weighted statistic such that the time a value persists is used to compute the statistical value. While the STATE statistic may appear to be the more complicated, it is the easiest to specify in SIMIO. For example, let's add a STATE statistic to determine the average number of Part 3's in the system first and then add a TALLY statistic for the average times in the system.

**Step 12:** Before creating a STATE statistic, it is necessary to create a state variable that will hold the values, for example, the number of Part 3's in the system. SIMIO will monitor the value of the state variable through a STATE statistic and weight its value according to the time a particular value persists throughout the simulation. Since the number of Part 3's in the system is a model-based characteristic and not associated with a particular entity, we need to define a model-based state variable. From the *Definitions→States* section, insert a new *Discrete Integer* variable with the name **GStaPart3NumberInSystem**, as seen in Figure 5.22



**Figure 5.22: Discrete, Integer, Model-based State Variable**

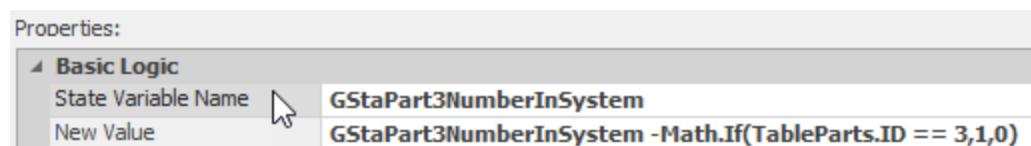
**Step 13:** The variable needs to be incremented when a Part 3 enters the system and then decremented when a Part 3 leaves the system. Let's use the *State Assignments* of the **SrcParts** for *Before Exiting*, employing the *Repeating Property Editor* to increment the new state variable, as shown in Figure 5.23.



**Figure 5.23: Incrementing the State Variable**

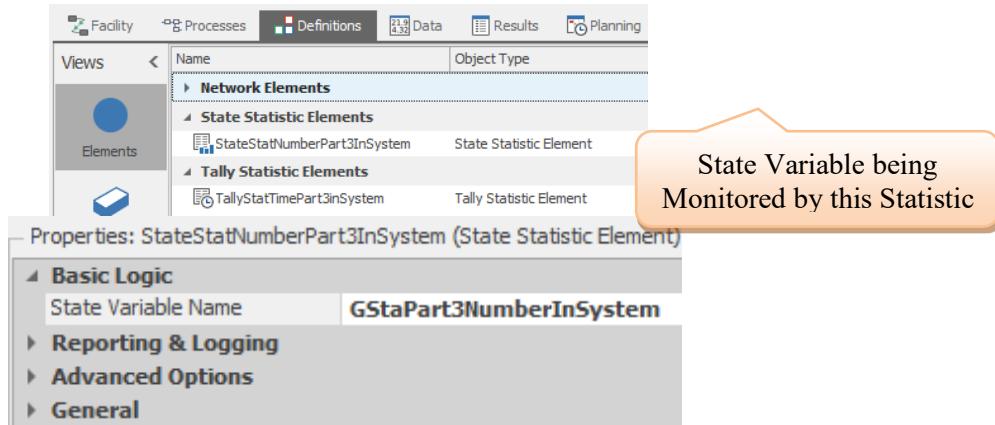
**Step 14:** The `Math.If()` function simply tests if the `TableParts.ID` has the value of "3," which refers to Part 3's, then adds "1" to the state variable, otherwise, the expression adds "0."

**Step 15:** In a similar fashion, the state variable can be decremented in the *State Assignments* of the **SnkPartsLeave** for *On Entering*, as shown in Figure 5.24.



**Figure 5.24: Decrementing the State Variable**

**Step 16:** Now all you need to do is to insert **STATESTATISTIC** from the *Definitions→Elements* section named **StateStatNumberPart3InSystem** that watches the state variable as seen Figure 5.25.

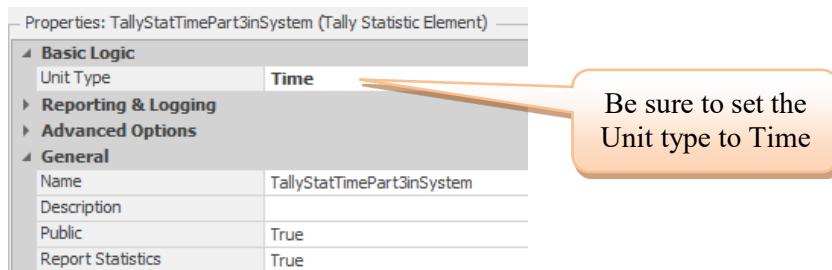


**Figure 5.25: Defining the State Statistic Element**

**Step 17:** Save your model and run your simulation for 40 hours.

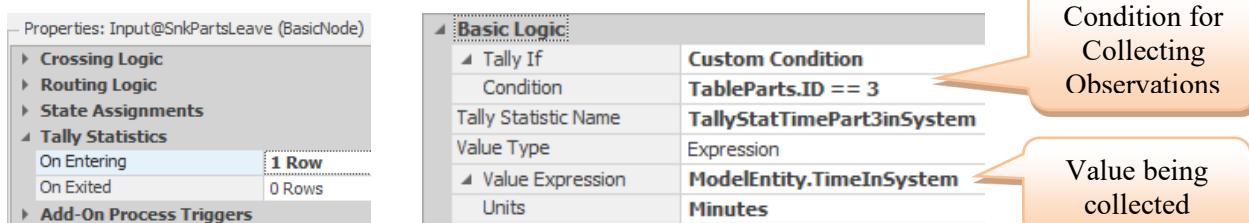
**Question 18:** What did you get for the average and the maximum number of Part 3's in the system?

**Step 18:** Now, we would like to collect time in system for the Part 3's, which is a TALLY statistic. The TALLY statistic needs to record individual observations. Tally statistics can be collected at any BASIC or TRANSFER node in their *Tally Statistics* property.<sup>54</sup> Unlike the STATE statistics, the TALLY statistic needs to be defined first. Using the *Definitions→Elements* section, insert a TALLY STATISTIC named **TallyStatTimePart3InSystem**. See Figure 5.26 for specifying the *Unit Type* property of “Time.”



**Figure 5.26: Defining the Tally Statistic Element**

**Step 19:** Select the **Input@SnkPartsLeave** node (i.e., the one attached to the **SINK**). Under the **Tally Statistics** Section, using the *On Entering* tally property, specify which observation (i.e., **ModelEntity.TimeInSystem**) to record using the *Repeating Property Editor*. Specify the collection of the tally statistic as shown in Figure 5.27.



**Figure 5.27: Specifying the Collection of Part 3 Time in the System**

<sup>54</sup> In later chapters we will use the *Tally* process steps to collect statistics anywhere.

**Step 20:** Note that a *Custom Condition* causes only Part 3's time in the system to be collected.

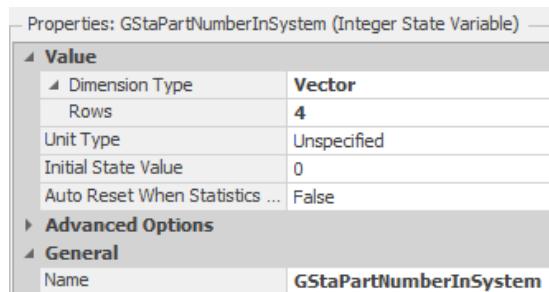
**Step 21:** Now, run your simulation for 40 hours.

**Question 19:** What did you get for the average, maximum, minimum, and number of observations for time in the system for Part 3?

## Part 5.7: Obtaining Statistics for All Part Types

Since statistics like time and number in the system would be useful for each part type, we will extend the statistics collection to all part types rather than just Part 3's.

**Step 1:** First, let's create the time in system statistics. Navigate to the *Definitions→States* section and modify the definition of **GStaPart3NumberInSystem**. Rename the variable to **GStaPartNumberInSystem**. Set the **Dimension Type** property of the state variable to a “Vector” with “4” rows, as shown in Figure 5.28. This will create a vector of four state variables, one for each part type.



**Figure 5.28: Defining the Number in System Vector**

**Step 2:** Next, change the state assignments at **SrcParts** and **SnkPartsLeave** to increment and decrement the number in the system for each part type, respectively, as shown in Figure 5.29. Note that the row (**TableParts.ID**) is used in the vector and [ ] to access the row in the vector in the expression.<sup>55</sup>

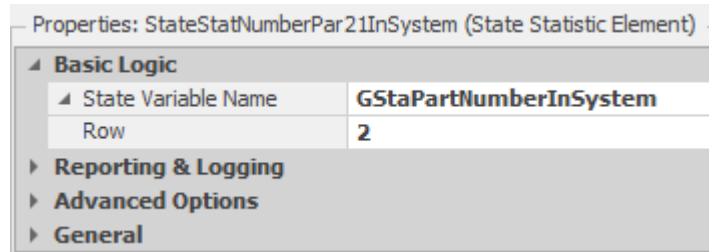
Basic Logic	
State Variable Name	GStaPartNumberInSystem
Row	TableParts.ID
New Value	GStaPartNumberInSystem[TableParts.ID] + 1

Basic Logic	
State Variable Name	GStaPartNumberInSystem
Row	TableParts.ID
New Value	GStaPartNumberInSystem[TableParts.ID] - 1

**Figure 5.29: Incrementing and Decrementing the Number in System**

**Step 3:** You can now define state statistics for all the parts. Under *Definitions→States*, create three more state statistics named **StateStatNumberPart1InSystem**, **StateStatNumberPart2InSystem**, and **StateStatNumberPart4InSystem**. You need to specify the state variable for each of the statistics using the previously defined vector state variable. This specification is the same for each statistic except for the row. The specification for Part 2 is shown in Figure 5.30. Modify all the four state statistics.

<sup>55</sup> The row index of the first row for Vectors, Matrices, and Data Tables is one while symbol indexes start at zero.



**Figure 5.30: Specification for StateStatNumberPart2InSystem**

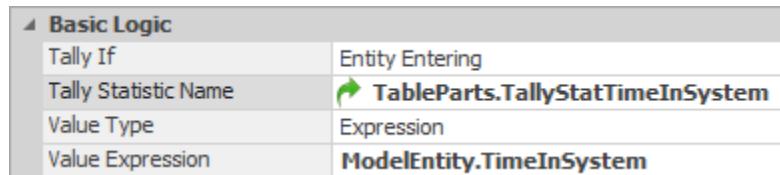
**Step 4:** Now define the TALLY statistics for the other three-part types (Parts 1, 2, and 4), just as was done for **TallyStateTimePart3InSystem** (see Figure 5.26 making sure the **Unit Type** property is equal to “Time.”

**Step 5:** Navigate to the *Data→Tables* section and select the **TableParts** DATA TABLE. Insert a new *Tally Statistic Element Reference* property named **TallyStatTimeinSystem**, allowing us to specify a tally statistic for each part. Therefore, each entity (i.e., part type) will know which statistic to update. There is little need to insert State statistics into the data table since we don’t reference them within the model – only the state variable that the state statistics monitor. Fill in the tables as shown in Figure 5.31.

Table Parts		Sequence Part		
	ID	Part Mix	Number to Arrive	Tally Stat Time In System
1	1	25	1	TallyStatTimePart1InSystem
2	2	35	1	TallyStatTimePart2InSystem
3	3	25	1	TallyStatTimePart3InSystem
4	4	15	5	TallyStatTimePart4InSystem

**Figure 5.31: Data Table Including Tally Statistics**

**Step 6:** Once the Table row reference has been assigned, each **MODELENTITY** (i.e., part) will have an associated TALLY statistic, eliminating the use of complicated deciding logic when determining which TALLY statistic to update. Select the **Input@SnkPartLeave** BASIC node in front of **SnkPartLeave**, and under the *Tally Statistics* entry, click on *On Exited*.<sup>56</sup> Use the *Tally Statistic Name* based on the row that has been assigned to the entity (i.e., **TableParts.TallyStatTimeInSystem**). The observation *Value* will be the model entity time in system value, as seen in Figure 5.32, to utilize the entity’s tally statistic.



**Figure 5.32: Time in System Tally Statistics**

**Step 7:** Now, re-run the simulation and examine the results.

---

<sup>56</sup> The *On Entering* could be used as well.

*Question 20:* Are the tally statistics on time in the system for each part type the same as the results from Part 5.4.?

---

*Question 21:* Do the state statistics for the number in the system for each part type match the results from Part 5.4.?

---

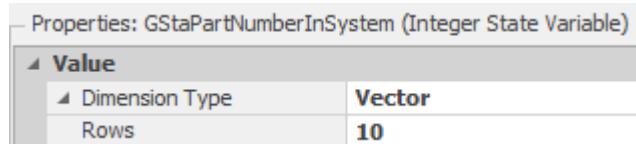
### Part 5.8: Automating the Creation of Elements for Statistics Collection

Adding intelligence to entities made it easy to model different routings and calculate time in the system and number in system statistics for each part type without creating multiple sink and source objects corresponding to the individual parts. However, we would still need to define the associated statistic objects and state variables. Such an approach becomes very cumbersome if we want to add many new part types.

An advantage of using related tables is that they may be linked (bound) directly to a spreadsheet containing many part types and sequences. Nevertheless, we may want statistics for time in the system (i.e., cycle times) and the number in the system for each part type. However, each part type will require a Tally and State statistic. Having to define each statistic could negate the advantages of using related tables as individual tally and state statistic elements need to be created, and the correct statistic must be collected for the particular part. However, SIMIO allows for the automatic creation of state variables and statistic elements.

**Step 1:** If you are using the model from the previous section, you should delete all the `TALLY` and `STATE STATISTIC` elements so these can be created automatically.

**Step 2:** Since the number of part types might change, the state variable `GStaPartNumberInSystem` must be modified to match the number of parts in the table each time. In our case, we must set it equal to the maximum possible number of part types in the system (e.g., 10).<sup>57</sup> The size of the state variable can be tied to the number of rows and columns of a data table. From the *Definitions→States* section, change the *Rows* property from “4” to “10”.



**Figure 5.33: Specifying the Size of the State Variable to Match the Table**

**Step 3:** Navigate to the *Data→Tables* section and select the `TableParts` DATA TABLE. Insert an *Element Reference Property→Tally Statistic* property named `TallyStatTimeinSystem`. Remove the tally statistics from that column, so it looks like Figure 5.34. Note that the tally statistics references are missing.

---

<sup>57</sup> “Matrix from Table” is another *Dimension Type* for state variables which will create a two dimensional matrix based on the number of rows and numeric columns of a table. Dynamically. However, it initializes the variable based on the table values.

Table Parts		Sequence Part		
	ID	Part Mix	Number to Arrive	Tally Stat Time In System
1	1	25	1	X
2	2	35	1	X
3	3	25	1	X
4	4	15	5	X

Figure 5.34: Adding a Tally Stat Reference Column

**Step 4:** Once one specifies entries (i.e., names), the elements will be created. The element's initial properties can be manually specified (e.g., *Category*, *Data Item*, *Unit Type*, etc.) from the *Definitions*→*Elements* section. However, since the number of part types can be dynamic, we do not want to have to specify these properties each time, which would defeat the benefit of automatic specification of the statistical elements. Therefore, the elements that are created can also have their properties initialized by specifying another column in the table as their values. For our problem, the category and data item classification of the TALLY statistics can be specified as we want it to be the same as other time in system statistics (i.e., “FlowTime” and “TimeInSystem”). Therefore, insert two standard *Property*→*String* property columns named **Category** and **DataItem**, as seen in Figure 5.35.

Table Parts		Sequence Part						
	ID	Part Mix	Number to Arrive	Tally Stat Time In System	Category	Data Item	Unit Type	State Stat Num In System
1	1	25	1	X	FlowTime	TimeInSystem	Time	X
2	2	35	1	X	FlowTime	TimeInSystem	Time	X
3	3	25	1	X	FlowTime	TimeInSystem	Time	X
4	4	15	5	X	FlowTime	TimeInSystem	Time	X

Figure 5.35: Adding Properties to the Statistical Elements

**Step 5:** For the *Unit Type* property of the TALLY statistic, insert a standard *Enumeration* property column named **UnitType** into **TableParts**, as was done in Figure 5.35. An enumeration is a property whose value is specified from a list of potential values. In this case, unit types can be “Unspecified,” “Time,” “Length,” etc. For the *Enumeration* column property, specify the *Enum Type* as “UnitType” to pull its values from the unit type enumeration list. Then, the *Default Value* should be “Time,” as seen in Figure 5.36.

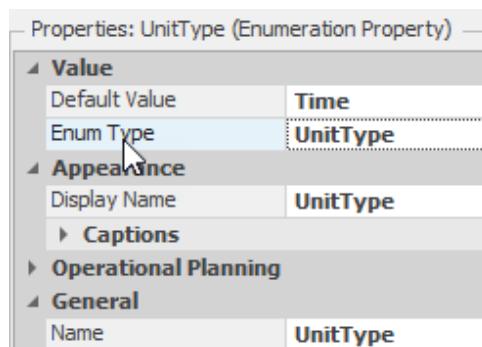


Figure 5.36: Setting up the Enumeration Property Column

**Step 6:** Next, insert a *State Statistic Element Reference* property named **StateStatNumInSystem**, as seen in Figure 5.35.

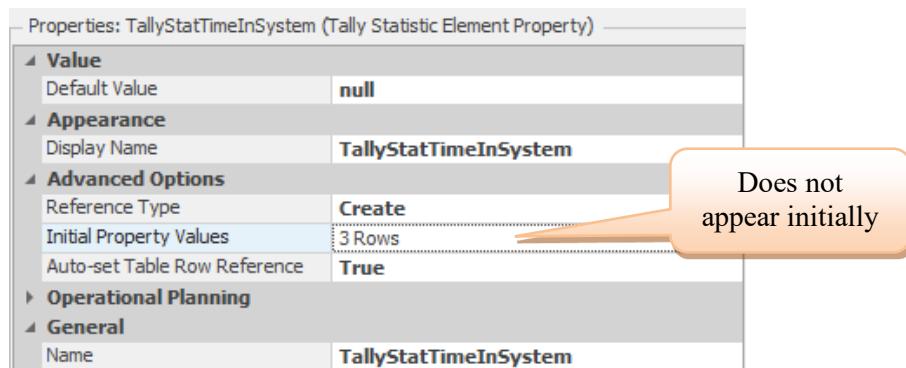
**Step 7:** Rather than creating each TALLY and STATE statistic to specify as entry values, SIMIO has the ability to automatically create elements (i.e., TALLY and STATE statistics, MATERIALS, etc.) specified from a table column. Therefore, each part row can have a specified TALLY statistic (i.e., name) that will automatically be created and used to keep track of its on-time in system statistics. By default, the *Element*

*Reference* property column will be of a “Reference” type, meaning the element would need to be already created to be specified as an entry in the table.

- Therefore, change the *Reference Type* property to “Create,” as shown in Table 5.8 and Figure 5.37.
- Set the *Auto-set Table Row Reference* to “True” as well.

**Table 5.8: Specifying Properties to Automatically Create the Element**

Property	Description and Value
<i>Reference Type</i>	“Reference” - It allows you to specify an element that has already been defined under the “Definitions” tab. “Create” - Entries in this column will now automatically create a new element of the specified type with the entry's name. Note that these elements will appear in the Definitions→Elements but cannot be deleted from here. Any name changes in either location (i.e., TABLE or Definitions→Elements) will be reflected in both.
<i>Initial Property Values</i>	This can be used to initialize the element based on constants or values in other table columns for that row.
<i>Auto-set Table Row Reference</i>	If “True,” then the element that is pointed to by each row will automatically be given a table reference set to that row. If you create and initialize the element based on other columns, this must be the case.



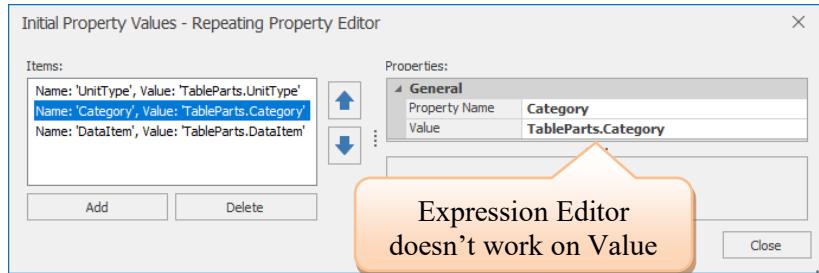
**Figure 5.37: Specifying Element Reference to Automatically Create the Element**

**Step 8:** After specifying the “Create,” the statistical elements will be created, but they will not have their properties referenced. To use the entries specified in these three new columns as the initial properties, select the **TallyStatTimeinSystem** column and set the *Auto-set Table Row Reference* to **True**, which assigns the current row to the newly created element. Next, click the *Repeating Property* button for the *Initial Property Values* property. Next, add the following three properties and values as specified in Table 5.9.<sup>58</sup> To specify the initial properties, use the repeating property window shown in Figure 5.38.

**Table 5.9: Specifying the Initial Properties**

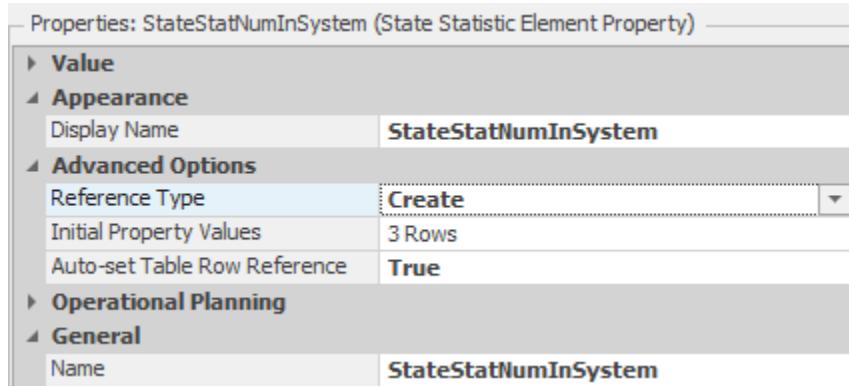
Property Name	Value
<i>Unit Type</i>	TableParts.UnitType
<i>Category</i>	TableParts.Category
<i>DataItem</i>	TableParts.DataItem

<sup>58</sup> You will need to type in the values directly as this is not an expression editor.



**Figure 5.38: Specifying the Initialization Properties of the TallyStat Reference Column**

**Step 9:** Select the **StateStatNumInSystem** and specify that this column will be created as well, as seen in Figure 5.39, making sure to set the *Auto-set Table Row Reference* to True.

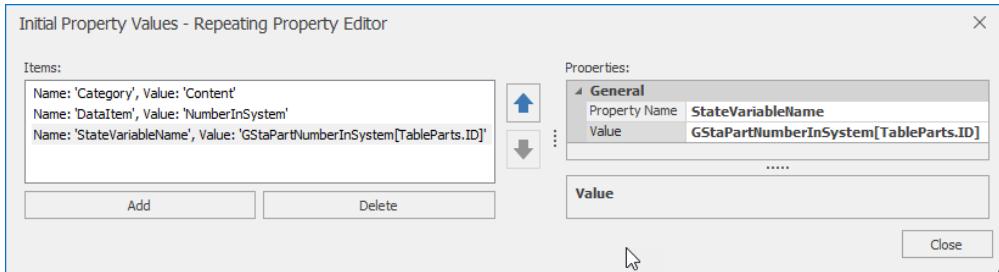


**Figure 5.39: Having the State Statistics to be Automatically Created**

**Step 10:** Next we need to initialize the new state statistics in a similar fashion to the Tally statistic column. However, we will not setup additional columns to specify the parameters. Set the “Category” and “DataItem” properties to string values, and the “StateVariableName” property will use the TableParts.ID to specify which state variable the statistic should monitor as seen in Table 5.10 and Figure 5.40.

**Table 5.10: Specifying the Initial Properties**

Property Name	Value
<i>StateVariableName</i>	GStaPartNumberInSystem[TableParts.ID]
<i>Category</i>	Content
<i>DataItem</i>	NumberInSystem



**Figure 5.40: Initial Parameters of the State Statistic Column**

**Step 11:** Add in the specifications for the properties of the four-part types as shown in Figure 5.41.

	ID	Part Mix	Number to Arrive	Tally Stat Time In System	Category	Data Item	Unit Type	State Stat Num In System
1	1	25		1	FlowTime	TimeInSystem	Time	
2	2	35		1	FlowTime	TimeInSystem	Time	
3	3	25		1	FlowTime	TimeInSystem	Time	
4	4	15		5	FlowTime	TimeInSystem	Time	

**Figure 5.41: Specifying the Properties of the Statistical Elements**

**Step 12:** You may now want to reset the *Reference Type* property back to “Reference.” Then, when you set the Reference Type property back to “Create”, the statistical elements will be automatically created as shown in the *Definition→Tally Statistical Elements (Auto-Created)* section. Note how each statistic takes its *UnitType*, *Category*, and *DataItem* from the DATA TABLE.

**Step 13:** Now, for every new row that is added, a new TALLY and STATE statistic will be created and initialized from the entry values of the *Category*, *DataItem*, and *UnitType* columns. For any rows that existed before the two columns were created, the values will not be initialized.<sup>59</sup> We can cut and paste values back and forth from SIMIO tables and Microsoft Excel™ spreadsheets. Select the entire table by highlighting the row selectors and then cut the rows (*Ctrl-x*). Next, paste them into a Microsoft Excel™ spreadsheet so you don’t have to retype all the information. You can also export the table to a CSV file from the *Content* tab under the *Table Tools* section. Modify the names of the tally and state statistics, categories, data items, and unit types, as seen in Figure 5.42.

	A	B	C	D	E	F	G	H
1		1	25	1 TallyStatP1	FlowTime	TimeInSystem	Time	StateStatP1NumInSystem
2		2	35	1 TallyStatP2	FlowTime	TimeInSystem	Time	StateStatP2NumInSystem
3		3	25	1 TallyStatP3	FlowTime	TimeInSystem	Time	StateStatP3NumInSystem
4		4	15	5 TallyStatP4	FlowTime	TimeInSystem	Time	StateStatP4NumInSystem

**Figure 5.42: Using Excel to Modify the Values**

**Step 14:** Copy the values in Excel and select the empty row in the SIMIO data table by clicking on the row selector. Then, paste the values into the table, which should now look like the one in Figure 5.34.



**Figure 5.43: Select the Empty Row of a Data Table to Paste all Values**

**Step 15:** Save and run the model for 40 hours.

*Question 22:* What is the average time in the system and the number in the system for each of the parts?

---

*Question 23:* Does the time in the system for Part 3 in TallyStatistic3 agree with the previous value computed?

---

<sup>59</sup> We can either modify the properties our self or we can delete the rows and recreate them. You can select the AutoCreate dropdown in each column value.

**Step 16:** Now add a fifth part type by creating a sequence and update only the data table (don't create any elements) by copying the fourth part information (i.e., have the same part mix and number of entities) but changing the **ID** to 5 and tally and state statistic column values, as well as copying the fourth part sequences for the fifth part.

*Question 24:* Did SIMIO create the tally statistical element automatically?

---

*Question 25:* When you execute the model, did the flow time and time in system statistics for the fifth part type appear with the proper unit type, category, and data item?

---

### Part 5.9: Commentary

- The *Start Date* property of the Work Schedule intuitively would be when the schedule will start. So, if one specified a future date, then nothing would happen until that date. That is not the case. The *Start Date* property represents the particular day the first day of the pattern will start. If this date is in the future compared to the start of the simulation, it will repeat backwards to the current date based on the pattern. For example, if we have a three-day pattern named Day 1, Day 2, and Day 3, and the work schedule start date is set to 09/17/23, the schedule will follow the Day 1 pattern for this date. If the simulation starts on 09/13/23, then the pattern for 09/13 is Day 3, 09/14 is Day 1, 09/15 is Day 2, 09/16 Day 3, which makes Day 1 on 09/17 and Day 2 on 09/18, etc. To have a work schedule not start until a particular day, exceptions to the workday have to be employed.
- Another way to look at SIMIO properties and states is that property values are established for each object in the Facility window at the beginning of the simulation execution and cannot be changed during “run-time.” In contrast, the value of states is also established at initialization, but it can be changed during “run-time.” Objects that are created during run-time, such as ENTITY, have their properties established during an instance of run-time, which is during the initialization of that instance. Otherwise, they cannot be changed during the simulation execution.
- Relational tables offer a tremendous advantage over other simulation languages in assigning one table and automatically assigning all records of related tables. We will explore this feature more in later chapters. In later chapters, we will demonstrate how you can bind data tables to Excel spreadsheets, which could facilitate the automatic creation of new parts and sequences.

# Chapter 6

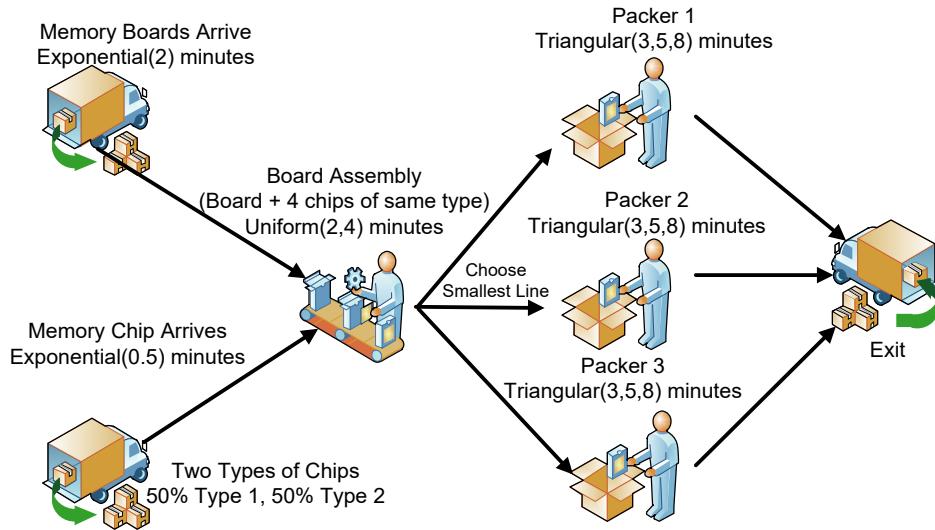
## Assembly and Packaging: Memory Chip Boards

---

A standard simulation environment in production is assembly and packaging. What makes these operations challenging is that entities are being combined together. The way they are combined can be complicated. Sometimes, the assembly is temporary, and the assembly (called a batch) must be separated later. These considerations cause us to have an interest in the `COMBINER` and `SEPARATOR` object.

### Part 6.1: Memory Board Assembly and Packing

Memory boards have memory chips inserted into them at an assembly operation, whose capacity is four. Each memory board requires four chips. Memory boards and individual memory chips arrive randomly, so the assembler waits for four chips of the same type and one board to be available before doing the assembly. After the board is assembled, it is sent to one of three packing stations, each of which has one worker. Boards are sent to the packing station with the smallest queue. Figure 6.1 depicts the memory board assembly and packing process.



**Figure 6.1: Memory Board Assembly and Packing**

The basic numerical characteristics of the assembly problem are found in Table 6.1.

**Table 6.1: Numerical Distributions of the Memory Board Assembly**

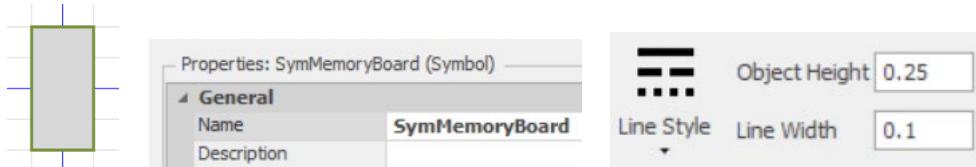
	<b>From</b>	<b>To</b>	<b>Travel time (minutes)</b>
<b>Travel Times</b>	Memory Board arrivals	Board Assembly	4 minutes
	Memory Chip arrivals	Board Assembly	5 minutes
	Board Assembly	Packer Station 1	Pert(10,12,15) minutes
	Board Assembly	Packer Station 2	Pert(5,7,10) minutes
	Board Assembly	Packer Station 3	Pert(4,5,7) minutes
	Any Packer Station	Exit	3.5 minutes

<b>Arrival Information</b>	Interarrival time for Memory Boards	Exponential(2) minutes
	Interarrival time for Memory Chips	Exponential(.5) minutes
	50% of Memory Chips are of type 1, and 50% are of type 2	Discrete
<b>Processing Times</b>	Board Assembly processing time	Uniform(2,4) minutes
	Packing time for all Packers	Triangular(3,5,8) minutes

**Step 1:** Create a new model that has two SOURCES named **SrcMemoryBoard** and **SrcMemoryChip** (i.e., one for boards and one for memory chips). Add two MODELENTITIES into the model, one for boards named **EntMemoryBoard** and one for memory chips named **EntMemoryChip**.<sup>60</sup> Ensure the *Entity Type* property of the two sources will create the appropriate entity type.

**Step 2:** Give the entity type associated with boards a different symbol, as seen in Figure 6.2. You can use one of the stock SIMIO symbols or download a symbol from the Trimble 3-D warehouse, as done in Chapter 2. You can also make your own symbols in the “Project Home” tab by clicking the “Create New Symbol” menu item from the *New Symbol* dropdown in the “Create” section. In the properties window, name the symbol **SymMemoryBoard**. When creating a new symbol, you should give the symbol “height” by specifying a value (in meters). In the case shown in Figure 6.2, a rectangle was created with a line width of 0.1 meters and a height of 0.25 meters<sup>61</sup>. The rectangle was given a fill color of grey with a line color of green. One should not worry about the exact specifications since the symbol can be adjusted manually.



**Figure 6.2: Symbol, Changing the Name, and Specifying a Height and Line Width**

**Step 3:** Select the “Model” in the [Navigation] panel to return back to the simulation model. Select the **EntMemoryBoard** model entity, click on the *APPLY SYMBOL* dropdown, and select the **SymMemoryBoard** symbol in the *PROJECT GRAPHICS* section to utilize our new symbol.

**Step 4:** For the memory chips, add an additional symbol<sup>62</sup> since we have two different types of memory chips. You should color the second symbol a different color (e.g., red) to distinguish them in the system.<sup>63</sup>

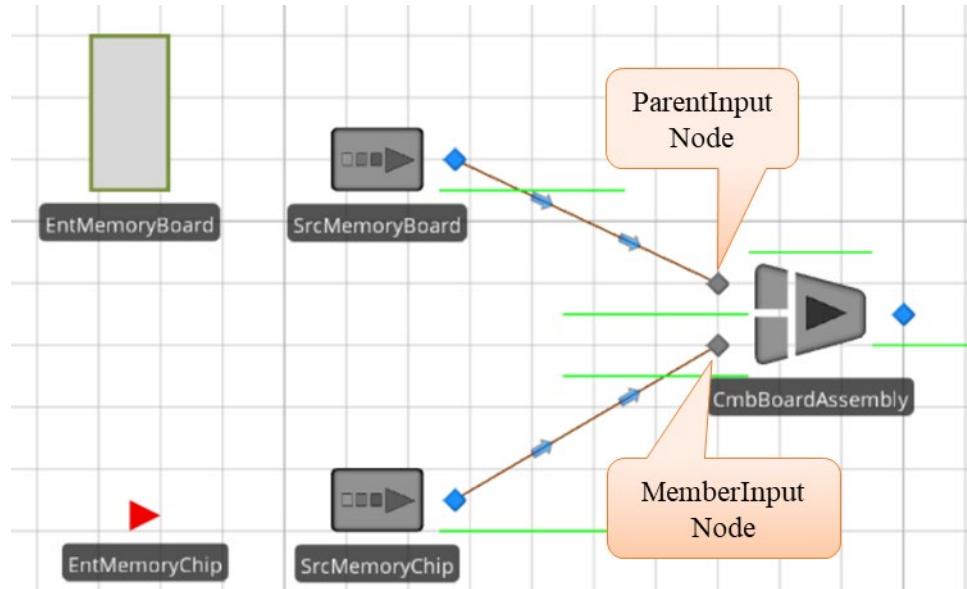
<sup>60</sup> To add entities, drag MODELENTITY from the [Project Library] panel.

<sup>61</sup> Note memory boards are not  $\frac{1}{4}$  meter thick but were made this thick to show a depth in 3-D view.

<sup>62</sup> Select the **EntMemoryChip** MODELENTITY and the click *Symbols*→*Additional Symbols*→*Add Additional Symbol* button.

<sup>63</sup> Select the 3-D view and color the sides of the new symbol as well.

**Step 5:** Insert a COMBINER named **CmbBoardAssembly** into the model for the assembly operation, as shown in Figure 6.3, which will batch/combine a parent entity object with a member entity object. Set the initial capacity of the **CmbBoardAssembly** to four and the processing time to Uniform(2,4) minutes. The combiner has two input nodes (i.e., a PARENTINPUT and a MEMBERINPUT). The parent object is the board, while the member object is the memory chip. Connect the two SOURCES via TIME PATHS to the appropriate nodes using the time specified in Table 6.1.



**Figure 6.3: Initial Memory Board Model**

**Step 6:** Objects have characteristics that describe and /or define their behavior or attributes. Properties and state variables are two very similar characteristics that allow the user to specify certain characteristics of the object (e.g., processing time for a SERVER object). Properties are characteristics that are set (i.e., initialized) when the object is created. For most objects, the initialization occurs at the beginning of the simulation and, therefore, cannot change during the run of the simulation. State variables are dynamic characteristics that allow their values to change during the simulation run and should be used when their values need to be altered during the simulation. We can modify two object definitions (i.e., MODEL and the MODELENTITY) as seen in the [Navigation] panel. Both the MODEL and MODELENTITY can have properties and state variables. However, the MODEL has only one set of properties and state variables that can be accessed (i.e., global scope) while each individual entity that is created will contain its own properties and state variables independent of the other entity instances.<sup>64</sup>

**Step 7:** To make the state variables clear when they are used in various places in a model, we prefix a MODELENTITY state variable with **ESta...** and a MODEL global state variable with **GSta....**

**Step 8:** Recall a board requires four identical memory chips. Therefore, the COMBINER should batch four members of the same type. Let's add an entity-based state variable to distinguish between the two types of memory. Select the MODELENTITY in the [Navigation] panel and then choose the “States” section from the “Definitions” tab. Since there are two distinct chip types, insert a new “Integer State Variable” named **EStaChipType**, as seen in Figure 6.4.

<sup>64</sup> Both the MODEL and MODELENTITY can have a state variable named **Color**. The only difference is how they are accessed. Specifying **Color** will access the variable of the MODEL while **ModelEntity.Color** will access the entity's variable.

Properties: EStaChipType (Integer State Variable)	
Value	
Dimension Type	Scalar
Unit Type	Unspecified
Initial State Value	0
Auto Reset When St...	False
Advanced Options	
General	
Name	EStaChipType
Description	
Public	True

Figure 6.4: Specifying the ModelEntity Integer State Variable

**Step 9:** Returning to the “Facility” tab of the **Model**, click on the **SrcMemoryChip** source. To assign the new **EStaChipType** variable, a *State Assignment, Before Exiting*, will be specified, which will make the assignment right before the entity leaves the source. Expand the *State Assignment* section. Then click on the box to the right of *Before Exiting*, which brings up the “Repeating Property Editor,” and add the following property.

- Since it is equally likely the chip will be type one or type two, a random discrete distribution will be used to assign the **EStaChipType** value a one or a two.

*State Variable Name:* ModelEntity.EStaChipType  
*New Value:* Random.Discrete(1, 0.5, 2, 1.0)<sup>65</sup>

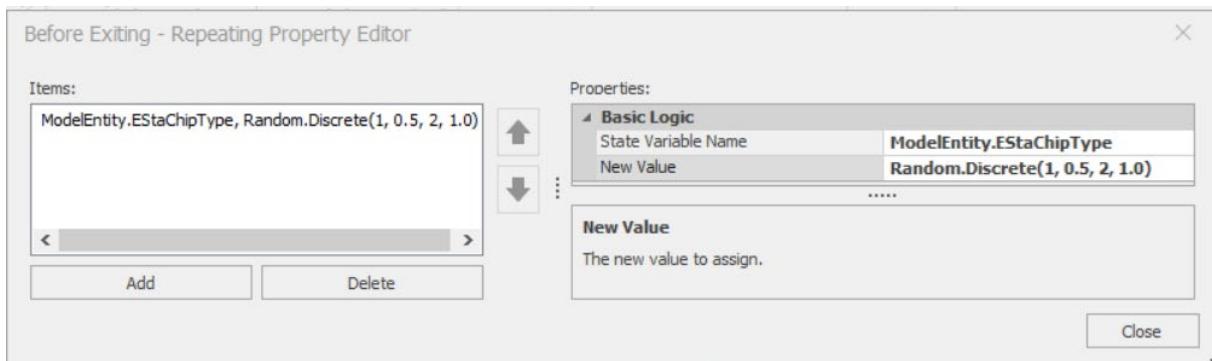


Figure 6.5: Setting the Entity’s Chip Type in a State Assignment

**Step 10:** Since the chip entities can be distinguished, batch four members (i.e., chips) at the **COMBINER** by specifying **Match Members** as the *Matching Rule* with **ModelEntity.EStaChipType** as the *Member Match Expression* property, as seen in Figure 6.6.<sup>66</sup> Then, expand Other Batching Options and change the value of *Must Batch Simultaneously* to **True**. All four chips must be at the combiner before batching begins.

<sup>65</sup> The *Discrete* distribution must use the appropriate cumulative probabilities.

<sup>66</sup> Combiners can match any entity, match members or match certain members and parents based on a criterion.

Properties: CmbBoardAssembly (Combiner)	
Batching Logic	
Batch Quantity	4
Matching Rule	Match Members
Member Match Expression	ModelEntity.EStaChipType
Batch Quantities (More)	0 Rows
Other Batching Options	
Parent Ranking Rule	First In First Out
Member Ranking Rule	First In First Out
Must Simultaneously Batch	True
Release Batch Early Trig...	0 Rows
Suspend Batching When ...	True

Figure 6.6: Specifying the Matching Logic for a Combiner

**Step 11:** Insert three SERVERS (i.e., SrvPacker1, SrvPacker2, and SrvPacker3) one for each packer with the appropriate processing times (see Table 6.1) and one SINK named SnkExit (see Figure 6.7). Utilize TIMEPATHS to connect the assembly to the packers and the packers to the exit. Specify the appropriate travel times specified in Table 6.1 for the time paths between the assembly and the packers, but do not specify the times between the packers and the exit quite yet.

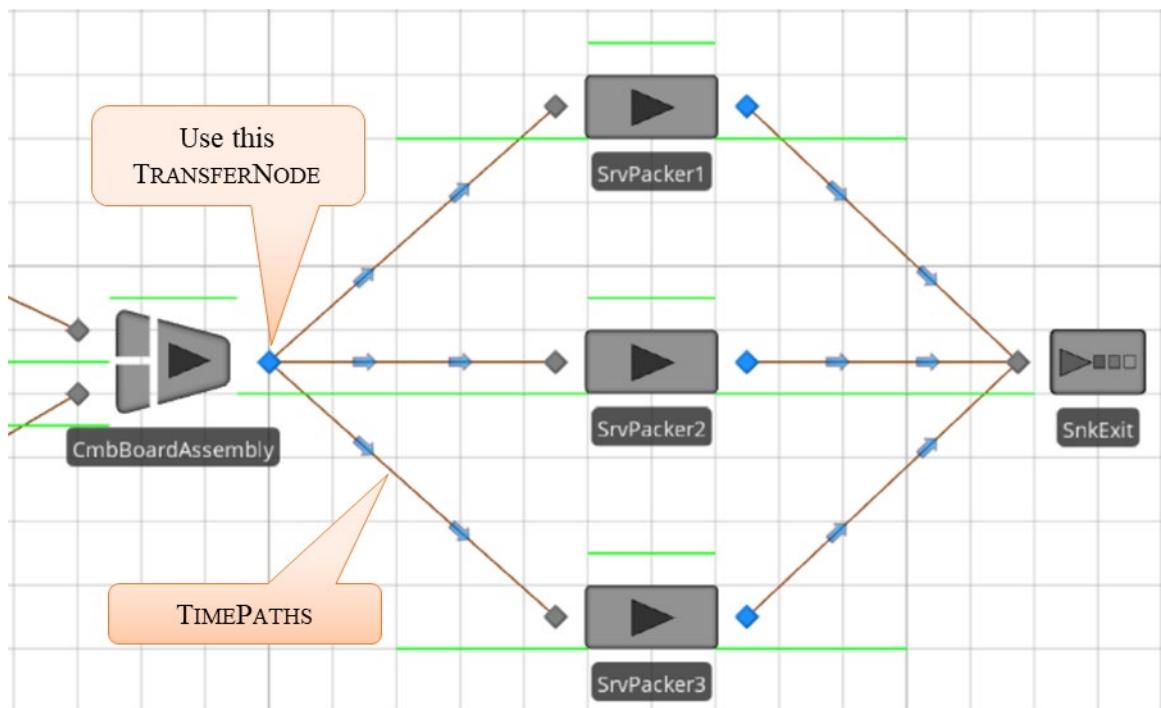
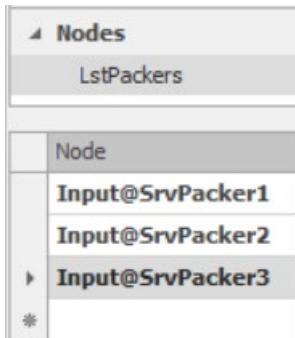


Figure 6.7: Packing Portion of the Model

**Step 12:** Recall the board assemblies need to choose the packer with the smallest number assigned to it. At the output TRANSFERNODE associated with the COMBINER, the branching decision needs to be specified using the *Entity Destination Type* property specification in the TRANSFERNODE.

**Step 13:** Once the board finishes assembly in the COMBINER, it can select one of the three destinations: **Input@SrvPacker1**, **Input@SrvPacker2**, or **Input@SrvPacker3**, depending on the number of boards already assigned/designated to each packer. Create a new *Node LIST* (see *Definitions*→*Lists* tab) named

**LstPackers**, beginning with **Input@SrvPacker1**, as seen in Figure 6.8. Lists are necessary whenever you need to choose from a set of objects (e.g., nodes, resources, etc.).



**Figure 6.8: Node List for Selecting the Best Packer**

**Step 14:** Now, at the TRANSERNODE, specify that entities should transfer to the destination according to the following properties, as seen in Table 6.2. Once you specify the *Selection Goal* property, you need to expand it to access the *Selection Expression*. We need an expression to evaluate the “shortest queue.” However, do we just mean the number in the respective queues? Probably not. How about including the number of entities being processed in the SERVERS and the number of entities traveling to the SERVER.

**Table 6.2: Properties for Selecting the Shortest Line**

Property	Value
<i>Entity Destination Type:</i>	Select from List
<i>Node List Name:</i>	<b>LstPackers</b>
<i>Selection Goal:</i>	<b>Smallest Value</b>
<i>Selection Expression:</i>	Candidate.Node.NumberTravelers.RoutingIn + Candidate.Server.InputBuffer.Contents.NumberWaiting + Candidate.Server.Processing.Contents.NumberWaiting
<i>Selection Condition:</i>	(Leave Blank <sup>67</sup> )

The CANDIDATE designation is a “wildcard” reference that takes on the identity of the object it references. Here, it refers to the node to which the entity is being routed and the particular INPUTBUFFER and PROCESSING queue of the SERVER at the end of the path. In this example, the word CANDIDATE is replaced by each node in the list and the expression is evaluated. In a later chapter, ASSOCIATEDSTATIONS functions will be used to simplify this expression. Leave the *Selection Condition* blank and leave the *Blocked Destination Rule as the default value*.

**Question 1:** How does the *Selection Expression* cause the entity’s alternative destinations to change relative to each destination?

---

**Step 15:** Since the travel time to exit is the same for all packers and we don’t expect to change it during the simulation, let’s specify the travel from the packers to the exit using a “Model Property” via the “Definitions” tab. Insert an *Expression* property from the “Standard Property” dropdown named

---

<sup>67</sup> It is very easy to specify the *Selection Goal* property as the *Selection condition*. The *selection condition* property is an expression that has to evaluate to “True” before the item can be selected from the list (e.g., SERVER has not failed) as a possible choice.

**TimeToExit** and give it a *Default Value* of 3.5 with *Unit Type of Time* with *Default Units* as **Minutes**. Since this is an “Expression” property, one could specify an expression for the time to exit (for example, `Random.Uniform(2, 10)`), while a numeric property only allows constant (i.e., 3.5).

**Step 16:** Now you can specify  **TimeToExit** as a referenced property for the *Travel Time* on each of the time paths to the exit<sup>68</sup>. Now, if you want to change the travel time on these three paths at the same time, only the new property has to be changed, or you want to experiment with the impact of this time. You can access the model properties by right-clicking on the **MODEL** in the [Navigation] and then expanding the “Controls” category to access our new property, as seen in Figure 6.9.<sup>69</sup>

**Step 17:** The **MODEL** properties become controls for the simulation. These can be changed at the beginning of any run. Furthermore, they become changeable scenario properties in experiments, as we saw earlier in Chapter 2.



**Figure 6.9: Changing the Properties of the Model**

**Step 18:** Save and run the model for 10 hours, answering the following questions.

*Question 2:* How long do the entities stay in the system (enter to leave)?

---

*Question 3:* What is the utilization of the **CmbBoardAssembly**?

---

*Question 4:* What is the utilization of each of the packers?

---

## Part 6.2: Making the Animation Reveal More Information

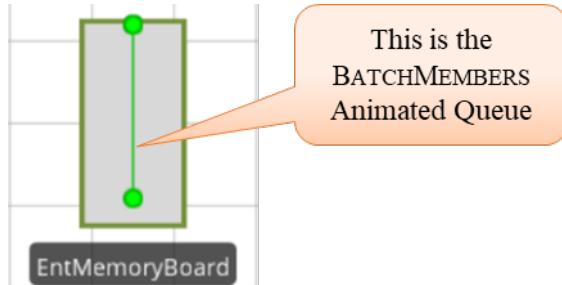
Let’s first fix the memory board animation so the “chips” look attached to the board.

**Step 1:** Select the **EntMemoryBoard** **MODELENTITY** and insert the **BATCHMEMBERS** queue from the “Attached Animation” in the “Queue” dropdown. Position it on top of the **EntMemoryBoard**, as seen in Figure 6.10.

---

<sup>68</sup> Select all three time paths and then right click on the *Travel Time* property and select the new reference property **TimeToExit**.

<sup>69</sup> Note that if a default value is not given, the simulation model may not run until you have set the property because it will be Null.



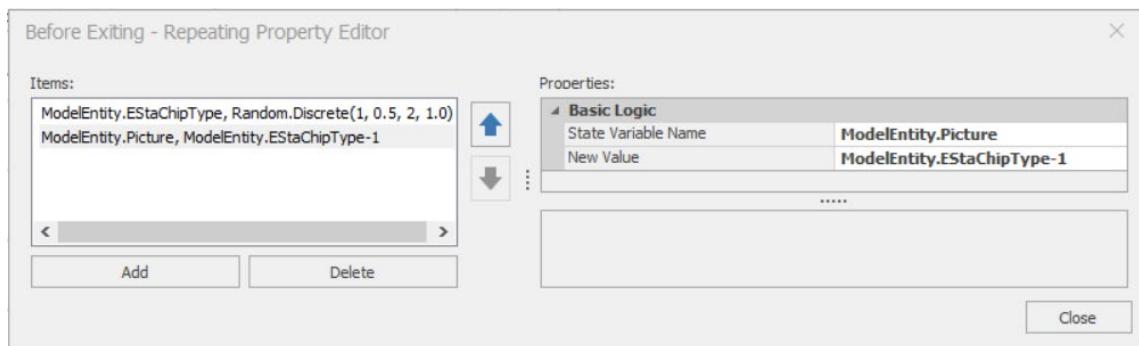
**Figure 6.10: Added Batch Queue**

To put an object on top of another, you need to select the object and hold down the *Shift* key to stack the queue on top of the memory board symbol. It's probably best to do this while switching back and forth from 2-D to 3-D.<sup>70</sup>

**Step 2:** Now, let's use the color of the memory chip symbol to designate the two different chip types. To do this, click on the **EntMemoryChip** ModelEntity and click on “Add Additional Symbol” in the “Additional Symbols” section of the “Symbols” tab. You can color one symbol red and the other green by simply selecting one of the “Active Symbols” and changing its color. Please note that these two symbols are numbered zero and one (**not one and two**).

**Step 3:** Next, we need to “assign” the colors to the different chip types. Select the **SrcMemoryChip** SOURCE. Expand *State Assignments* and click on the box to the right of *Before Exiting* since we want to assign the color before the entity leaves the source. This brings up the “Repeating Property Editor.” Add an additional property with the following characteristics.

*State Variable Name:* ModelEntity.Picture  
*New Value:* ModelEntity.EStaChipType-1<sup>71</sup>



**Figure 6.11: Changing the Chip Entity to have a Different Color**

Unlike a property, a “state variable” can be changed throughout the simulation. Here, the picture is a SIMIO-defined characteristic of the entity (in some simulation languages, it is called an attribute of the entity).

<sup>70</sup> Remember that hitting the “h” key in the model window brings up instructions for moving around in 2D and 3D and also in manipulating objects. Hitting the “h” key again removes the instructions.

<sup>71</sup> Recall the picture symbols start at zero which is why we need to subtract one.

*Question 5:* Why do we subtract one from the **entity's EStaChipType** property?

---

**Step 4:** Run the model to be sure the animation behaves as expected.

*Question 6:* Do the entities change the color to correspond to their chip type?

---

**Step 5:** Suppose now we want the symbol leaving the packing station to represent a “package.” There are several ways one might do this. One easy way is to make a *State Assignment* by selecting *Before Exiting* at the Packing stations. Now, at one of the Packing stations, bring up the *Repeating Property Editor* and, this time, add:

*State Variable Name:* ModelEntity.Size.Height  
*New Value:* 1

**Step 6:** You may need to experiment with the value. The idea is to change the memory board’s “height” so that it encases the “chips” so they cannot be seen (i.e., hides the batch members inside the board).

**Step 7:** Run the model and adjust the “height” so it shows a “package” leaving the packer. You will need to add this change in “height” at each packer station when the entity exits.

**Step 8:** Look at the model in 3-D and notice the changes.

**Step 9:** Save the model as Chapter 6.2.spfx and run the model for 10 hours.

*Question 7:* How long are the entities in the system (enter to leave)?

---

*Question 8:* What is the utilization of the **CmbBoardAssembly**?

---

*Question 9:* What is the utilization of each of the packers?

---

*Question 10:* What is the average number in the **MemberInputBuffer**?

---

*Question 11:* What is the average number in the **ParentInputBuffer**?

---

*Question 12:* What is the average number in the processing queue in the **CmbBoardAssembly** queue?

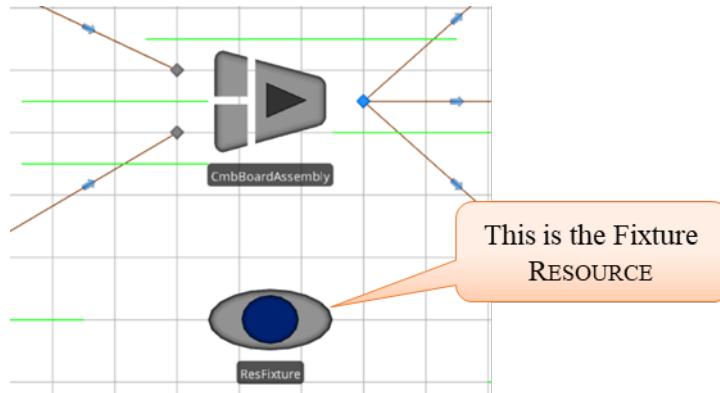
---

### Part 6.3: Embellishment: Other Resource Needs

Sometimes, capacity is not just limited to objects like SERVERS and COMBINERS. The limitation may not even be fixed at a particular location. For example, suppose the board assembly requires a particular fixture to assemble the chips to the board. Furthermore, that fixture is used to transfer the assembly to packing. For each assembly, the fixture must be first obtained from the packing operation, which takes about three minutes, and we will relax

the assumption. Let's assume there are ten identical fixtures available for assembly-packing, and once they are released at the pacing, they can be used immediately at the assembly.

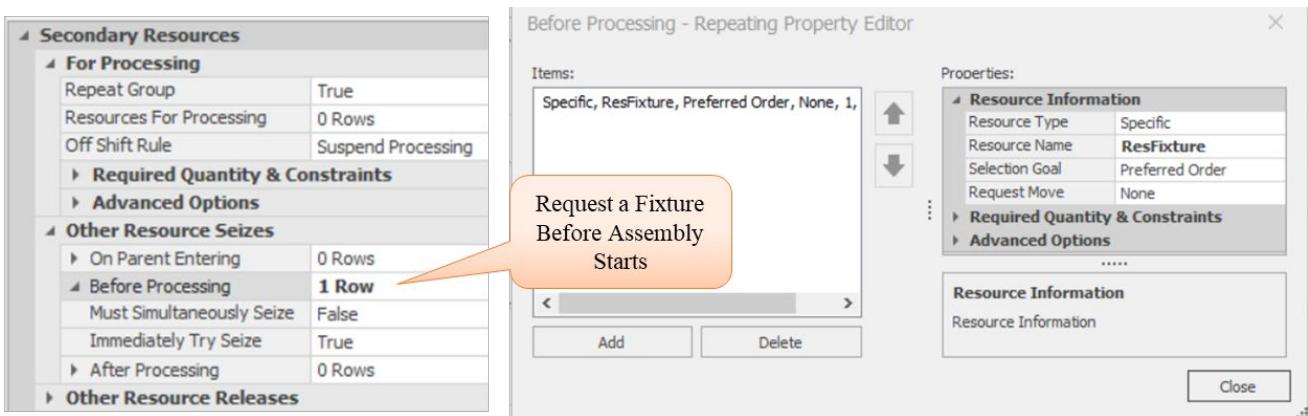
**Step 1:** First, drag and drop a RESOURCE object<sup>72</sup> onto the modeling canvas, maybe just below the COMBINER, as shown in Figure 6.12. Name the resource **ResFixture** and specify an *Initial Capacity* of 10 (we won't be changing its capacity in this case).<sup>73</sup>



**Figure 6.12: Adding the Fixture Resource**

Next, logic needs to be specified on how this RESOURCE is to be utilized. The fixture will be “seized” just before the assembly operation begins and then released just after the packing operation finishes.

**Step 2:** For the **CmbBoardAssembly** COMBINER, select the *Secondary Resources*, the *Other Resource Seizes*, and the *Before Processing*, as seen in Figure 6.13. Click the ellipses box to bring up the *Repeating Property Editor* for seizing resources and request one unit of the **ResFixture** resource capacity to be seized. We won't worry about which fixture is seized since they are identical.



**Figure 6.13: Seizing One Unit of Resource Capacity**

<sup>72</sup> Later chapters will go in more detail on the RESOURCE object.

<sup>73</sup> Note, a RESOURCE is fixed object and cannot travel throughout the network. Other types of resources (i.e., WORKER, VEHICLE, and ENTITY) can be dynamic and move through out a network.

**Step 3:** When the package is finished, the fixture needs to be released for the next assembly. This process has to be done for each of the three packers. For each packer, select the *Secondary Resources* and then the *Other Resource Releases*, and then the *After Processing*. Click the ellipses box to bring up the *Repeating Property Editor* for releasing resources. As shown in Figure 6.14, one unit of the **ResFixture** resource capacity will be released.

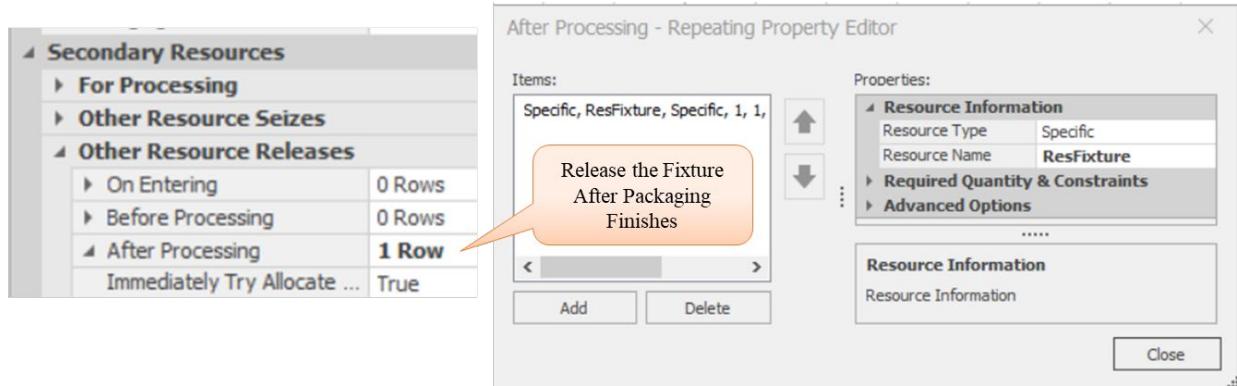


Figure 6.14: Releasing the Fixture

**Step 4:** In order to “see” how the available capacity of the resource changes during the simulation, add a *Status Label* from the *Animation* section in the ribbon to make sure no objects are selected.<sup>74</sup> When the status label is selected, the cursor can be used to draw a box onto the modeling canvas, as shown in Figure 6.15. Note that the expression gives the remaining capacity of **ResFixture**.

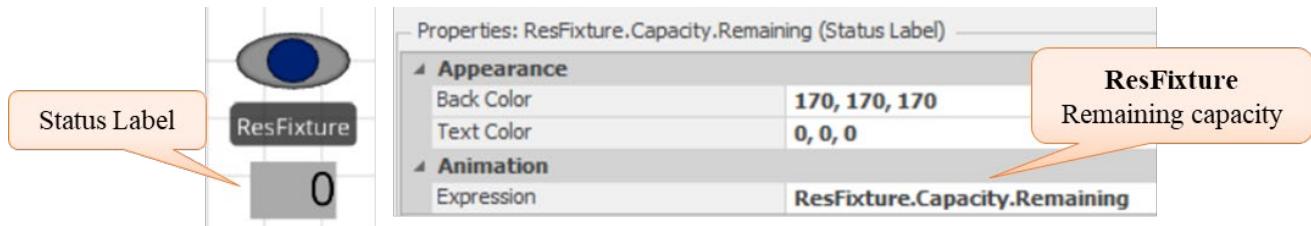


Figure 6.15: Adding a Status Label for Remaining Capacity

**Step 5:** Run the model and record the time in system for boards and chips.

**Step 6:** Make the capacity of the **ResFixture** equal to “20” and re-run the model.

**Question 13:** What is the difference in the time in system for boards and chips for ten versus 20 fixtures?

---

**Question 14:** Does it make sense to purchase ten additional fixtures?

---

<sup>74</sup> If we had selected the **ResFixture** first before inserting the status label, the label would be attached to the resource and therefore the expression would be just `Capacity.Remaining`. If we moved the Resource in the model the status label would move as well.

## Part 6.4: Changing Processing Time as a Function of the Size of the Queue

Suppose you find out that **SrvPacker1** changes its processing time depending on the number currently waiting (i.e., the size of the queue). Specifically, the packer is only 80% as efficient when there is zero in the input buffer queue, 90% when there is one, and 100% for all others.

**Step 1:** You need an “efficiency” table now to determine the processing time. This table can be modeled as a “Discrete Lookup” table in SIMIO (i.e.,  $f(x)$ ). From the “Data” tab, add a “Lookup Table” whose values are 0.8 for 0, 0.9 for 1, 1 for 2, and 1 for 100, as shown in Figure 6.16. Name the table **LookupEfficiency**.

X	Y
0	0.8
1	0.9
2	1
100	1

Figure 6.16: Lookup Table for Efficiency Calculation

Remember, the table gets interpolated for values between the discrete points. To access the efficiency based on the number waiting in the queue at the **SrvPacker1**, we will use the following expression: `LookupEfficiency[SrvPacker1.InputBuffer.Contents.NumberWaiting]`. Notice that to access rows (i.e., indexing) into tables and arrays, SIMIO uses the `[]` notation rather than `()`.

**Step 2:** We are planning on using the expression more than once in our model. Since the expression is very long and complicated, we will utilize SIMIO’s function expression creator to alleviate mistakes.<sup>75</sup> From the “Definitions” tab under the *Functions ( $f(x)$ )* section, insert a new FUNCTION named **FuncEfficiency**. Set the *Expression* property to the appropriate value, as seen in Figure 6.17.

Properties: FuncEfficiency (Function)	
Value	
Expression	<code>LookupEfficiency[SrvPacker1.InputBuffer.Contents.NumberWaiting]</code>
Return Type	Any
Unit Type	Unspecified
General	
Name	FuncEfficiency
Description	
Public	True

Figure 6.17: Specifying a SIMIO Function Expression

<sup>75</sup> If we were to change the expression, it will only need to be modified in one place rather than in every location we used the direct expression.

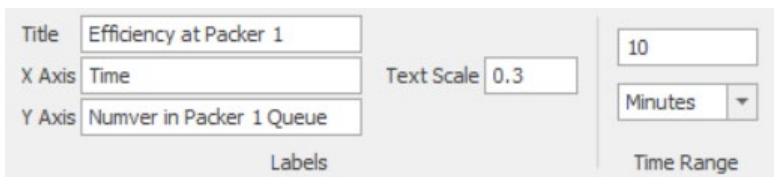
**Step 3:** Now, in the *Processing Time* property for the **SrvPacker1**, change the expression such that the Triangular distribution is divided by the efficiency function to get the actual processing time. The new expression is defined as the following.

Random.Triangular(3, 5, 8) / FuncEfficiency

**Question 15:** Why do we divide the original processing time by the efficiency?

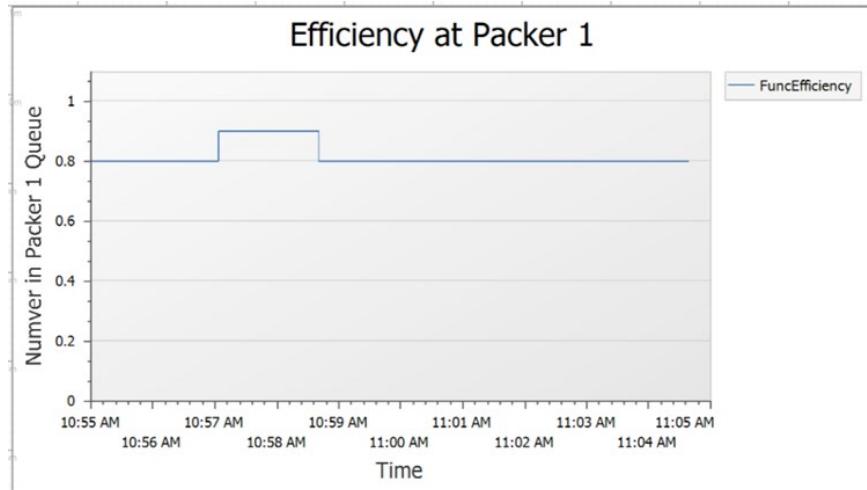
---

**Step 4:** We want to be sure our “efficiency” function is working correctly. To do that, we will use a SIMIO “Time Plot,” which you will find under the “*Animation*” tab in the ribbon. Use the following specifications for the plot, which are in the “Labels” and “Time Range” sections of the “*Appearance*” tab.

<b>Title:</b>	<b>Efficiency at Packer 1</b>	
<b>X Axis:</b>	<b>Time</b>	
<b>Y Axis:</b>	<b>Number in Packer 1 Queue</b>	
<b>Text Scale:</b>	<b>1</b>	
<b>Time Range:</b>	<b>10 Minutes</b>	

**Figure 6.18: Setting the Properties of the Status Plot**

Also, add the *Expression* property of the Status Plot that will be drawn, which is *FuncEfficiency* where the plot is given in Figure 6.19.



**Figure 6.19: Efficiency Plot**

**Question 16:** Does it appear that the efficiency function is working?

---

**Question 17:** What happens to the chips when they are assembled at the combiner?

---

**Question 18:** Can you visually distinguish between the two chip types?

---

## Part 6.5: Creating Statistics

Here, the SIMIO “automatic” statistics collection process may not provide the statistics that interest you. For example, we might want to know the time the boards and chips get from entry to exit, or we might want to know the average efficiency for **SrvPacker1**. Time in system or time in a “sub-system” is an example of “observation-based” statistics. Recall this type of statistic, which is referred to in SIMIO as a “Tally Statistic.” In the case of efficiency, its value is changing (discretely) with respect to time, and it is an example of what is generally referred to as a “time-persistent” or “time-weighted” statistic. In SIMIO, this type of statistic is a “State Statistic,” and since efficiency changes discretely, it is referred to as a “Discrete State Statistic.”<sup>76</sup>

**Step 1:** Consider first the “state” statistic “efficiency.” We need to define a model-based (global) state variable for efficiency in collecting statistics on this characteristic. From the “Definitions” tab, select “States,” and in the “Discrete” section, click on “Real” to define a new state variable named **GStaPacker1Efficiency**.

**Question 19:** Notice there are seven general discrete types of “state” variables, but only the “Discrete→Real” one is relevant here. What is there about the efficiency change that is “discrete” and “real”?

---

**Question 20:** What are some of the other “value types” within the “Discrete” variable type?

---

**Step 2:** To ensure that **StaPacker1Efficiency** is correctly monitored for its value by SIMIO, let’s modify the “State Assignments” in the **SrvPacker1 SERVER**. In particular, we will make the assignment *On Entering* the server and use the following assignments.

<i>State Variable:</i>	<b>GStaPacker1Efficiency</b>
<i>New Value:</i>	<b>FuncEfficiency</b>

**Step 3:** Finally (this is the last step in collecting a state statistic), we need to define a “State Statistic.” New statistics can be defined from the “Definitions” tab but in the “Elements” section. Elements represent special objects that add modeling flexibility. Let’s call the new statistic the **StateStatPacker1EfficiencyLevel** and use **GStaPacker1Efficiency** as the *State Variable Name*, as seen in Figure 6.20.

---

<sup>76</sup> Don’t confuse a “state variable” with a “state statistic”.

Properties: StateStatPacker1EfficiencyLevel (State Statistic Element)	
Basic Logic	
State Variable Name	GStaPacker1Efficiency
Reporting & Logging	
Data Source	
Category	Efficiency
Data Item	Number
Log Observations	False
Advanced Options	
General	
Name	StateStatPacker1EfficiencyLevel
Description	
Public	True
Report Statistics	True

Set Results  
“Category” and  
“Data Item”

**Figure 6.20: Setting a State Statistic**

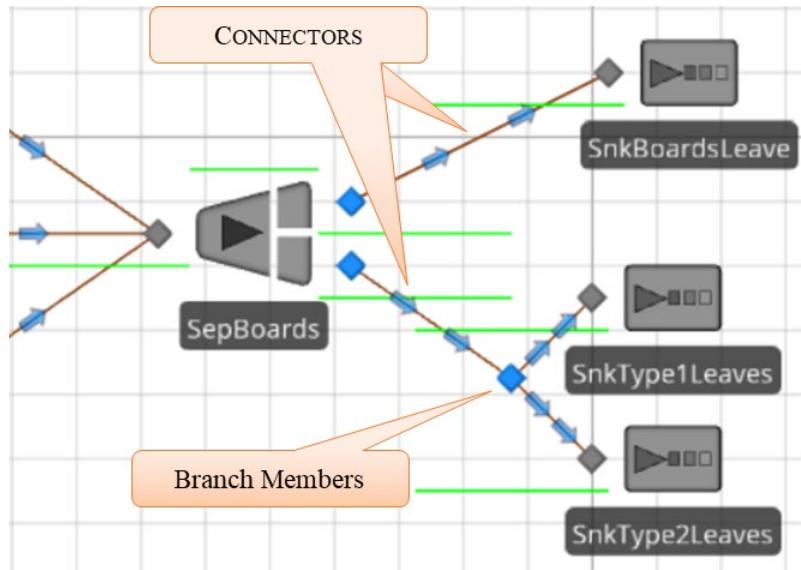
**Step 4:** Save and run the simulation model for ten hours. Filter the results of the “Object Type” to “Model” to see the new statistic.

*Question 21:* What did you get for the average **StateStatPacker1EfficiencyLevel** ?

---

**Step 5:** Next, let’s obtain the “time in system” statistics for the boards and chips to go from entry to exit. Although you might think you need to give some special instructions to SIMIO, this is a case where we can take advantage of the way SIMIO provides statistics. Recall that we automatically get “time in system” statistics on the entities exiting through a sink. We will exploit this modeling approach.

**Step 6:** To obtain the statistics on the boards and chips, we can separate them and send them through separate exits (i.e., SINKS) – and do all this without changing the fundamental statistics. Our approach is shown below in Figure 6.21.



**Figure 6.21: Separating Entities**

The approach will substitute a SEPARATOR (from the [Standard Library]) for the original SINK and then branch the “parent” and “members” output to the appropriate SINK. One can either delete the SINK and add a SEPARATOR or right-click on the sink and select *Convert To Type→SEPARATOR*. The members get further branched to separate SINKS using a TRANSFERNODE. CONNECTORS are used to connect objects, so no time is taken.

**Step 7:** Again, using time paths, connects the packers to the input of the SEPARATOR, specifying the *TimetoExit* property as the travel time. The branching of members uses “Link Weight.” The *Selection Weight* property for the link connecting to the SnkType1Leaves SINK is ModelEntity.EStaChipType==1 while the *Selection Weight* property for the connector to the SnkType2Leaves SINK is ModelEntity.EStaChipType==2.

**Step 8:** Run the model for 10 hours and answer the following questions.

*Question 22:* How long are boards in the system from the time they enter to when they leave?

---

*Question 23:* How long are “red” chips and “green” chips in the system (enter to leave)?

---

*Question 24:* What is the average utilization of Packer 1 over the 10 hours?

---

## Part 6.6: Commentary

- Good animation can add to modeling insight and provide a communication vehicle.

# Chapter 7

## Using SIMIO Processes

---

Most simulation languages hide the details of executing the processes associated with an event. Generally, SIMIO has taken care of performing these steps, and we have been somewhat ignorant of what specifically happens unless you have looked at the trace. Also, it is usually difficult to look “under the covers” in most simulation languages and add your own features to the simulation model.

An innovative feature of SIMIO is the capability to modify an object’s behavior through the use of SIMIO “Processes.” In essence, you can determine some new things that will happen based on current conditions. SIMIO processes offer a wide range of modeling opportunities, which will be employed in this and later chapters. Processes are very flexible and can model many varied behaviors beyond those intrinsic to the object (such as a SERVER). Processes allow you to extend and expand the modeling capability of native SIMIO. There are two basic considerations when using and developing processes.

1. “When” is the process invoked or executed?
2. “How” do you write a process to produce the desired behavior?

By now, you realize that a simulation is executed by moving from event to event. SIMIO has defined certain events from which you can respond with a process to change the model behavior. Processes are a sequence of simulation “steps” typically invoked by actions of objects within the simulation or by events whose response you write. Writing a process is similar to creating a stylized programming flowchart, except that the components of the flowchart are logical simulation “steps” such as *Assign*, *Decide*, *Tally*, *Transfer*, *Seize*, *Delay*, *Release*, and so forth. The actual execution of the process is performed by a “token.” The TOKEN is a SIMIO object that can have state variables, but it is not the same as an entity. The tokens execute the steps of a process on behalf of an object, typically the object whose behavior is being modified by the process or where the process resides (i.e., PARENT OBJECT) or the object causing the modification (i.e., ASSOCIATED OBJECT). The relationship among the token, objects, and process is shown in Figure 7.1.

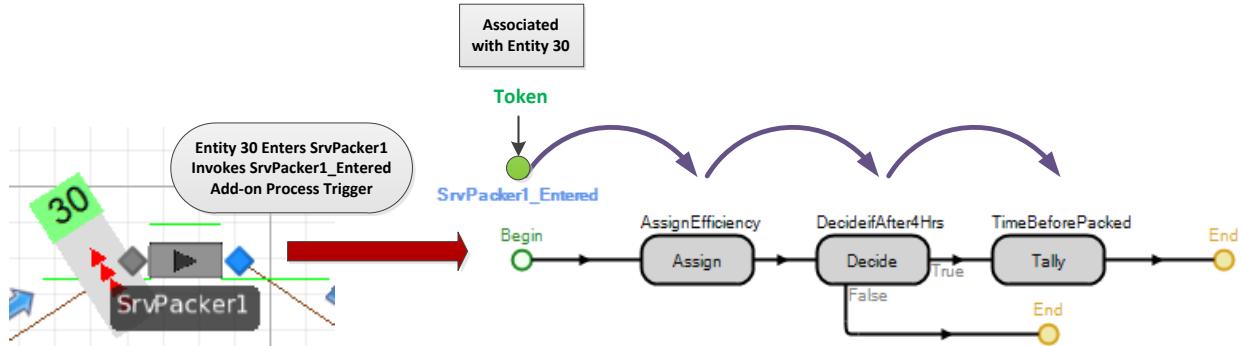
### Part 7.1: The Add-On Process

We will initially focus on the SIMIO “Add-on Process” triggers. These are events triggered by actions of objects but mostly by entities to change the behavior or cause an action for an object that is needed for your particular model. In the case of an Add-on process, the PARENT OBJECT is primarily the main MODEL since the Add-on processes are created within the main Model and are viewed and edited in the “Processes” tab of the MODEL. Often, the ASSOCIATED OBJECT is the MODELENTITY that triggers the particular process to be executed.

Figure 7.1 shows an example of the “Entered” Add-on process trigger. When Entity30 enters the SrvPacker1 server, the SrvPacker1\_Entered process is triggered. At this point, a TOKEN associated with Entity30 is created at the *Begin* endpoint and travels step to step in zero time (e.g., the TOKEN is currently at the beginning, getting ready to execute the *Assign* step) until it reaches the *End* endpoint at which time it is destroyed. In this example, the ASSOCIATED OBJECT is Entity30, and the PARENT OBJECT is the main Model. The TOKEN is executing the steps on behalf of Entity30.<sup>77</sup>

---

<sup>77</sup> You can have more than one TOKEN associated with the same MODELENTITY which will be seen in later chapters.



**Figure 7.1: Example Explaining Entity Triggering an Add-on Process**

## Part 7.2: The Add-On Process Triggers: Illustrating “Assign” Step

Almost all SIMIO objects have “*Add-On Process Triggers*.” These triggers cause external processes to be invoked as part of other processes. So, as a part of the execution of the object, arbitrary “processes” of your design can also be executed, which helps change/expand the underlying object’s behavior (i.e., SERVER) without having to look inside the object, which will be explored in a later chapter.

In Chapter 6, the characteristics of the entities were changed. In one case, the entity types (i.e., the memory chips) were colored, and in the second case, the memory board entity’s height was changed. In both cases, we used “state variables” whose values were changed using the “*State Assignments*” properties of the SOURCE and SERVER objects. You will see that there are only a few opportunities within an object to make state assignments, while there are many more Add-On Process Triggers. Furthermore, the “*State Assignments*” can only assign state variable values, whereas a process can incorporate more logic than just simple assignments.

**Step 1:** Let’s see how to replicate the “*State Assignments*” using Add-On Process Triggers. Bring up the model from the prior chapter. First, remove the “*State Assignment*” from the **SrcMemoryChip** SOURCE by deleting it within the *Repeating Property Editor*.

**Step 2:** Make sure the **SrcMemoryChip** is selected, and click on the “*Add-On Process Triggers*” menu to expand it.

**Question 1:** What *Add-On Process Triggers* are available in the SOURCE object?

---

**Question 2:** When are “*State Assignments*” performed on entities made in the SOURCE object?

---

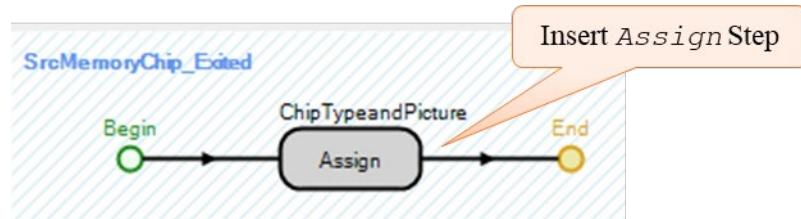
**Step 3:** Choose the “*Exited*” Add-On Process trigger. Double-clicking on this trigger automatically creates a new process named **SrcMemoryChip\_Exited**, which then selects the “*Processes*” tab. You will see a “flowchart” with a “*Begin*” and an “*End*,” as seen in Figure 7.2. By clicking on one of the process *Steps*, you can insert it into the flowchart, which will adjust itself automatically.

**Question 3:** Name a few of the steps that you might use to create a process.

---

*Question 4:* Why did you choose these?

**Step 4:** Insert an *Assign* step as shown in Figure 7.2.

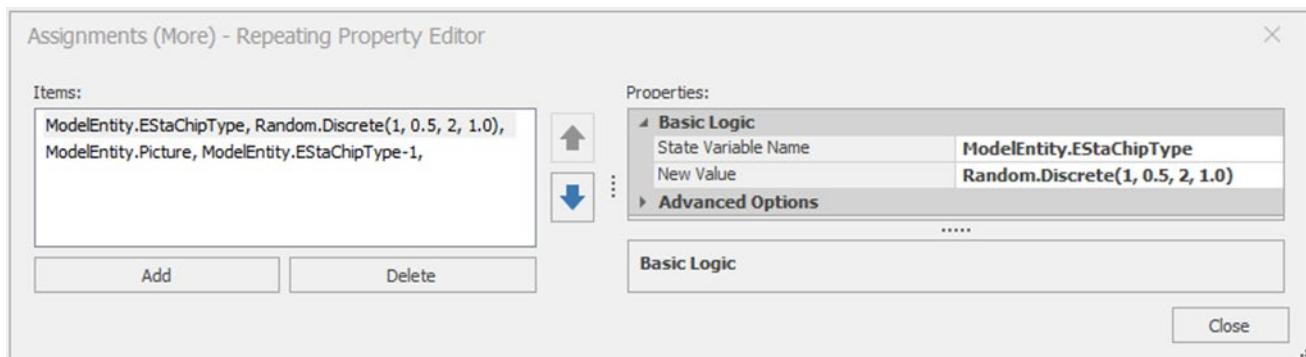


**Figure 7.2: Inserting the Assign Step**

**Step 5:** Under the properties window, you will see a place to put the *State Variable Name* and its *New Value*. You can only assign one state variable. However, you can also invoke the *Repeating property Editor* by clicking beside the “*Assignments (More)*.” Add the following two assignments using the latter approach, as seen in Figure 7.3.<sup>78</sup> It should look very familiar as the “*State Assignments*” utilize an *Assign* step internally.<sup>79</sup>

**Step 6:** *State Variable Name*: ModelEntity.EStaChipType  
*New Value*: Random.Discrete(1, 0.5, 2, 1.0)

**Step 7:** *State Variable Name*: ModelEntity.Picture  
*New Value*: ModelEntity.EStaChipType-1



**Figure 7.3: Repeating Property Editor for Assignment Process Step**

**Step 8:** Recall from the previous chapter, we calculated statistics on the efficiency of packer one. The efficiency was used to change the processing time of the packer based on the number in the queue. The state variable was updated when the entity entered the packer, including the current entity, rather than when the entity entered processing, where the efficiency value is used. The reason for not calculating the statistic correctly is there are only two default places for assignments (i.e., entering or exiting the server), but processes allow more flexibility. To fix the issue, delete the state assignment from the “*On Entering*” at **SrvPacker1**. Create a “*Processing*” Add-On Process Trigger which will calculate the efficiency right before the processing starts. Insert an *Assign* process step that does the following – see Figure 7.4.<sup>80</sup>

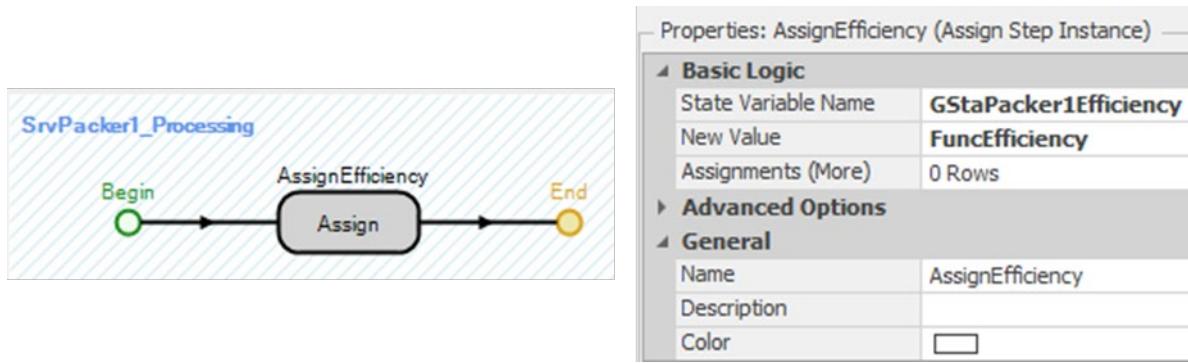
<sup>78</sup> When performing more than one assignment, it is recommended you perform all of them in the *Assignments (More)* to avoid issues of duplication, etc. The single assignment is performed before the *Assignments (More)*.

<sup>79</sup> In the later chapters on sub-classing we will explore how SIMIO does the assignments when we take apart some of the basic objects.

<sup>80</sup> Utilize the single assignment for this exercise.

*State Variable Name:* GStaPacker1Efficiency

*New Value:* FuncEfficiency



**Figure 7.4: Recreating State Efficiency Assignments**

**Step 9:** Save the model and run this model to make sure it replicates the behavior from the previous chapter.

**Question 5:** What was the average efficiency, and was it much different from the previous chapter?

### Part 7.3: Creating an Independent “Reusable” Process

Creating a “generic” process that can be used in several places within a model is useful. In the model from the prior chapter, the assignment to change the height of the board to make the entity look like a package was done at each packer station. Of course, this approach could be duplicated by creating an “Exited” add-on process trigger for each packer. However, the same exact logic of assigning the height has to go in each of the three “Exited” add-on process triggers. Instead, let’s create a “common” process that we can refer to for each packer rather than creating the same one for each packer. This type of process is called a “reusable” process and allows for modification in one place rather than three.

**Step 1:** To do this, select the “Processes” tab and click the “Create Process” button naming the new process **Packaged**. Next, add an *Assign* step to its flowchart with the following assignment.

*State Variable Name:* ModelEntity.Size.Height

*New Value:* 1 (Meters)

**Step 2:** Delete the *Before Exiting “State Assignments”* at each of the packer stations.

**Step 3:** For each of the packer object’s “Exited” Add-On Process Trigger, instead of creating a new one; just select the **Packaged** process from the drop-down list. Again, select all three packers to change them all at once, as seen in Figure 7.5.

Add-On Process Triggers	
Run Initialized	
Run Ending	
Entered	
Before Processing	
Processing	SrvPacker1_Processing
After Processing	
Exited	Packaged
Failed	
Repaired	
Evaluating Seize Request	
On Shift	
Off Shift	

**Figure 7.5: Utilizing a Common Process in all Three Packers**

**Question 6:** What is the advantage of having one process and referring to it at each packing operation over having duplicated processes at each packing station?

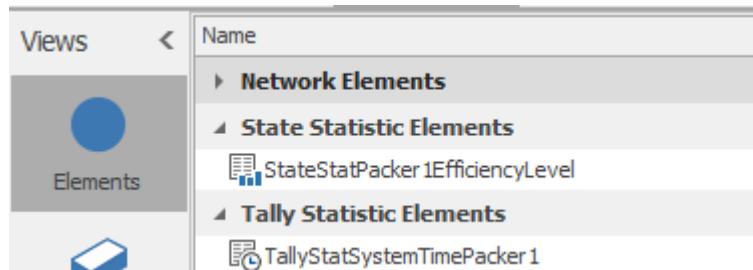
---

**Step 4:** Run this model and make sure it replicates the previous behavior.

#### Part 7.4: Collecting Tally Statistics

Recall that observation-based statistics are referred to as **TALLY** statistics in SIMIO. These statistics are gathered as observations as the simulation executes. Examples would be queuing times, flow times, cycle times, and time in system. For the most part, these are statistics whose observations are intervals of time. Previously, **TALLY** statistics collection was limited to **BASICNODES**, **TRANSFERNODES**, or the default results. Now, we want to greatly broaden the possible collection points, namely in any **PROCESS**.

**Step 1:** Let's consider collecting **TALLY** statistics on the time boards, and chips spend in the system, from arrival time to the time they start to be packed at Packer1 only. To accomplish this addition, we need to first define a new “*Element*.<sup>81</sup> When working with **TALLY** statistics, we define them before determining their collection method. Under the “*Definitions→Elements*” section, click on “*Tally Statistic*” in the ribbon to insert a new statistic named **TallySystemTimePacker1**, as seen in Figure 7.6.



**Figure 7.6: Specifying a new Tally Statistic in the “Elements” Section**

**Step 2:** Be sure to make the “*Unit Type*” for the statistic “*Time*.”

---

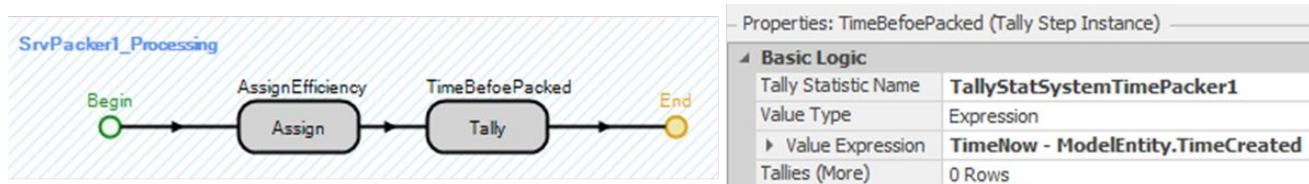
<sup>81</sup> Recall, *Elements* are behaviors of objects that can change state over time.

**Step 3:** Next, we need to define the collection process (i.e., logging each observation). This is done by inserting a **Tally** step into the **SrvPacker1\_Processing** add-on process trigger, as seen in Figure 7.7.

*Tally Statistic Name: TallyStatSystemTimePacker1*

*Value Type: Expression*

*Value: TimeNow - ModelEntity.TimeCreated*<sup>82</sup>



**Figure 7.7: Using a **Tally** Step to Track Time before Packing Station 1**

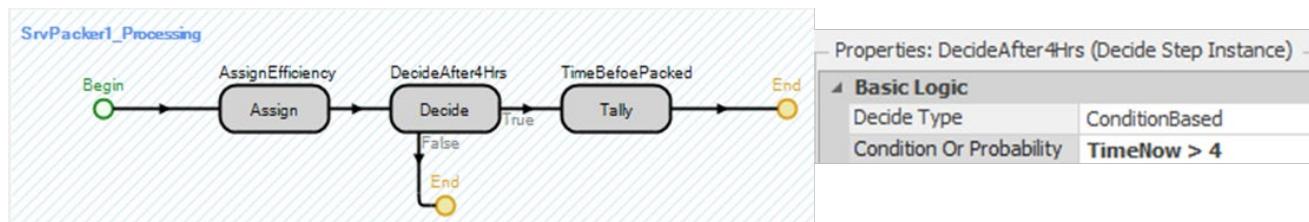
**Question 7:** Could the *Value* expression simply be *ModelEntity.TimeInSystem*?

**Step 4:** Save and run the model, observing the results.

**Question 8:** What did you get for the cycle time for boards and chips from the time they arrive to the time they start to be packed at Packer1?

**Step 5:** Suppose now we want only the time in system for assemblies as they start to be packed at **SrvPacker1** after the first four hours of the simulation? We need to modify our process so only observations of boards are collected after the first four hours.

**Step 6:** Modify our “**SrvPacker1\_Processing**” add-on process by adding a **Decide** step before the **Tally**, as shown in Figure 7.8. The **Decide** step is similar to an If statement in Excel, where when the condition is true, the process (i.e., TOKEN) will continue via the “True” branch; otherwise, it will follow the “False” branch. The **Decide Type** property should be “ConditionBased” with the **Expression** property set to *TimeNow > 4*.



**Figure 7.8: Adding a Decide Step**

**Step 7:** Save and run the model.

**Question 9:** What did you get for the arrival time for boards and chips from when they arrive to when they start to be packed at **SrvPacker1**?

<sup>82</sup> *TimeNow* is the current time of the simulation and *TimeCreated* function returns the time the entity was created.

**Step 8:** It is also possible to use process steps to replace the modeling changes we made to the original model in order to produce time in system statistics for each chip type (i.e., red and green) and board. The previous approach placed additional objects (i.e., SINKS) in the model, which were not part of the real system and might confuse management (or anyone else) when looking at the model.

- First, insert the original **SnkExit (SINK)**.
- Delete all the additions (i.e., the SEPARATOR and the three SINKS) when the original **SnkExit SINK** was replaced.<sup>83</sup>
- Next, connect the three packing stations back to the SINK as before, making sure to specify the **TimeToExit** property as the *Travel Time*.

**Step 9:** Next, let's define three new TALLY STATISTICS (remember these are “*Elements*”) named **TallyStatBoardTimeInSystem**, **TallyStatType1TimeInSystem**, and **TallyStatType2TimeInSystem**. Change the *Unit Type* to **Time**. To make these easier to find in the output, under “Results Classification,” set the “Category” to **FlowTime** and the “Data Item” to **TimeInSystem**, as seen in Figure 7.9.

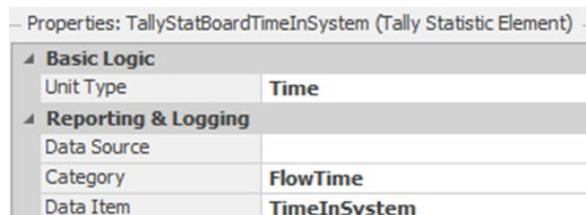


Figure 7.9: Specifying Time in System Tally Stats

**Step 10:** We will now create an *Add-On Process* in the SINK using the “*Destroying Entity*” add-on process trigger. This process is called **SnkExit\_DestroyingEntity** and its process is shown in Figure 7.10.<sup>84</sup>

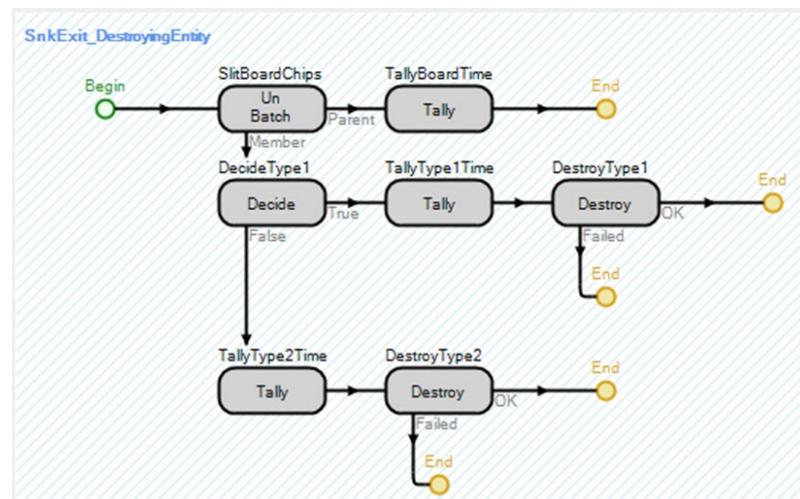


Figure 7.10: Destroying Entity Process (Tallying Statistics)

<sup>83</sup> When you delete a node, all the connections to it disappear. You can quickly connect nodes by holding the *Shift* and *Ctrl* key down, and clicking on the origin node and dragging a connector to the destination. You will need to select the type of connection (here we used TIMEPATHS whose *TravelTime* is a reference “**TimeToExit**”).

<sup>84</sup> Double clicking on the *Exited* Add-on process trigger property will automatically create the process as well as take you to the “*Processes*” tab.

Note the *UnBatch* step comes from the “All Steps (A-Z)” category in the steps panel. The *UnBatch* works much like the **SEPARATOR** object, except it breaks up the parent and the members separately in the batch and sends them along the correct branches of the step.<sup>85</sup> Here, two tokens are active on behalf of the same associated object: one **TOKEN** executes on behalf of the **PARENT**, and the second **TOKEN** executes on behalf of the **MEMBER**<sup>86</sup>.

- The *Decide* step is conditional, based on the condition `ModelEntity.EStaChipType==1`.
- The *Tally* steps simply record `ModelEntity.TimeInSystem` at each step for each of the three appropriate **TALLY** statistics.
- The *Destroy* steps are needed to eliminate the “unbatched” members. The Parent will be destroyed at the sink, but the members are unattached and will remain in the model<sup>87</sup> unless removed.

*Question 10:* What are the advantages and disadvantages of this approach versus the original model, where the symbolic model was changed to gather the different statistics?

---

*Question 11:* What are some of the other steps that appear in “All Steps (A-Z)” that don’t appear in the “Common Steps”?

---

**Step 11:** Save and rerun the model, observing what happens when compared to the previous model.

*Question 12:* How long are the boards in the system (enter to leave)?

---

*Question 13:* How long are the “red” chips in the system (enter to leave)?

---

*Question 14:* How long are the “green” chips in the system (enter to leave)?

---

**Step 12:** The results should be comparable to what you obtained in the previous chapter.

*Question 15:* Since using processes duplicated what we did with the original model, which approach is the easiest for you? <sup>88</sup>

---

---

<sup>85</sup> Internally the **SEPARATOR** utilizes an *UnBatch* step to separate the parent and member entities and then sends them out the appropriate output nodes.

<sup>86</sup> In this case the second token will cause four tallies – one for each of the four chips. Furthermore, the execution of the “member” occurs before the execution of the “parent” from an *Unbatch*.

<sup>87</sup> The members are unbatched into “FreeSpace”, which designates nowhere in particular, but they take up computer memory.

<sup>88</sup> People often try to avoid using processes at first, especially if they can model without them (this is part of the “different thinking” required in SIMIO). However you will find that the time invested in learning to use processes will pay off handsomely as you encounter more complex systems. The remaining chapters will illustrate why processes are so essential.

## Part 7.5: Manipulating Resources

Previously, we used a fixture from the assembly through the packing. A RESOURCE object with a capacity of ten was used to model the availability of the fixtures. We employed the SERVER's built in "Secondary Resources" to seize one unit of resource capacity before processing at the board assembly. We did not release that capacity until the entity had finished processing at the packer station.

Unfortunately, if additional resources are needed at a SERVER or COMBINER, the places where capacity may be "seized" or "released" are limited. Furthermore, all that can be done at those locations is a "seize" or "release" (seize and release if used only for "processing"). Recall: before the fixture can be used, it must be retrieved from the packaging area before processing and then torn down after packaging (i.e., packaging separated from the memory board). One approach would be to add server objects for these activities and place them into the model. These additional objects again begin to clutter the model with substantial objects that perform only simple functions that processes can handle, as well as model these situations more easily.

Whenever we are modeling the use of capacity within an object, three steps are often employed (i.e., *Seize*, *Delay*, and *Release*). Using these steps, you "seize," "delay" for some time, and then "release" capacity. When you seize capacity, it is important to know how many units of a resource are being seized. It is possible to seize units of multiple resources at the same time.

Now, we want to delay a Uniform(1,3) minutes after seizing the resource to set up and retrieve it and then delay a Uniform(2,4) minutes for a teardown separation process before releasing the resource. Before we simply choose one of the Add-on processes, let's consider the choices related to the processing, as seen in Table 7.1.

**Table 7.1: Few of the SERVER's Add-on Process Triggers**

Add-on Process Trigger	Description
Before Processing	Occurs when an entity has been allocated SERVER capacity, but before entering (i.e., or ending transfer into) the processing station.
Processing	Occurs when an entity has been allocated SERVER capacity, has ended transfer into the SERVER's processing station, and is now about to start the processing time.
After Processing	Occurs when an entity has completed the processing time and is about to attempt to exit from the SERVER's processing station.
Exited	Occurs when an entity has exited the SERVER object

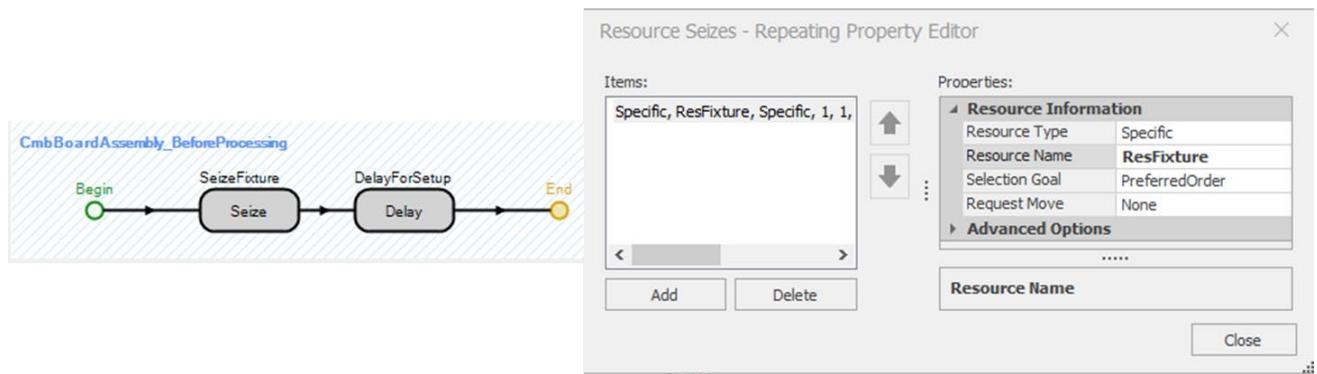
We will (somewhat arbitrarily) choose *Before Processing* for the place to seize the fixture and *After Processing* for when it should be released. Note that the time associated with setting up and tearing down the fixture contributes to the time being processed at the objects.

**Step 1:** First, remove from the **CmbBoardAssembly** the "Secondary Resources→Other Resource Seizes/Before Processing" the seize of one unit of capacity of **ResFixture** by deleting the entry in the *Repeating Property Editor*.

**Step 2:** Next, remove from each of the Packing stations the "Secondary Resources→Other Resource Releases/After Processing" the release of one unit of capacity of **ResFixture** by deleting the entry in the *Repeating Property Editor*.

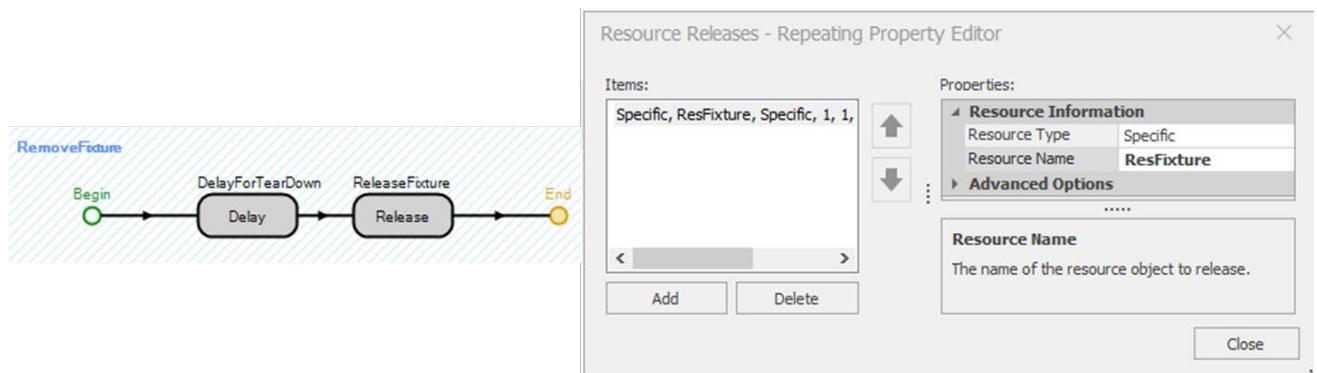
**Step 3:** For the **CmbBoardAssembly** COMBINER, create the "Before Processing" add-on process trigger named **CmbBoardAssembly\_BeforeProcessing**. This trigger is executed right before the COMBINER is ready to start processing the assembly operation. The *Seize* step will be used to specify that one unit of

ResFixture capacity is needed at processing, as shown in Figure 7.11. Once a fixture has been seized, a Uniform(1,3) minute Delay will be the time to set up and retrieve the fixture.



**Figure 7.11: Seizing One Unit of Fixture Capacity and Delaying**

**Step 4:** When the package is finished, the fixture must be removed and released for the next assembly. This process has to be done for each of the three packers, so it will be easier to create a reusable process that is invoked when the board exits the packing stations. From the “Processes” tab, click the “Create Process” button to name the new process **RemoveFixture**. The *Delay* step and the *Release* step are shown in Figure 7.12. The delay will take a Uniform(2,4) minutes to tear down the fixture before the fixture is released.



**Figure 7.12: Teardown and Release the Fixture**

**Step 5:** Now select the *After Processing* Add-On trigger for each of the Packers and select RemoveFixture from the drop-down list.

**Step 6:** Run the model for 8 hours.

**Question 16:** After looking at the results, what concerns do you have?

---

**Question 17:** What changes to the system could we suggest to make it more reasonable?

---

## Part 7.6: Tokenized Processes

The previous section showed how the *exact* same process could be reused in multiple locations. However, if the process was not completely identical (i.e., the teardown time depended on which packer), a process may be

given arguments through the tokens that execute them. These processes execute similarly to arguments in a programming language subroutine or subprogram. They parameterize the process or what we call a “tokenized” process since it requires custom tokens.

The need for a tokenized process occurs whenever you have several almost identical processes. For example, the time in system statistics for assemblies as they start to be packed at **SrvPacker1** after the first four hours of the simulation was needed. The “solution” was to create a process specialized to computing the time in system to that packer. If we wanted that same time in system for **SrvPacker2** and **SrvPacker3**, we would need to create two new processes, one for each packer and two more statistics. We could probably copy the first packer process for the second two and edit them accordingly. Of course, every time one copies and edits something, there is an increased chance of making an error. Also, if a change or additional behavior is needed later, all three processes would need to be updated.

**Step 1:** Let’s instead consider using a *Tokenized Process* that passes in the particular statistic to record the operation. First, define new time in system (TALLY) statistical elements for Packer2 and Packer3 as seen in Figure 7.13, and do not forget to change the *Unit Type* property to “Time.”

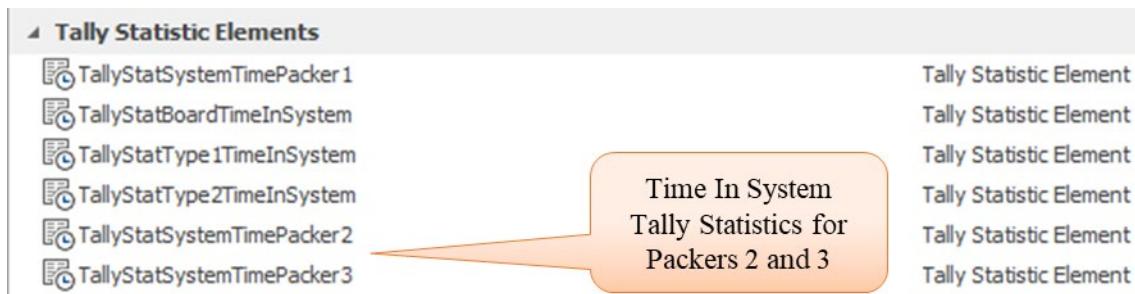


Figure 7.13: Tally Statistics for Packers 2 and 3

**Step 2:** Recall a TOKEN is created in order to execute the steps of a process. Therefore, parameters are passed to the process via a custom TOKEN. From the “Definitions→Token” section, add a custom TOKEN named **TknPacker**. This TOKEN needs an additional “Element Reference” state variables<sup>89</sup> named **TStaTallyStat**, to reference the correct statistic that will be used to store a parameter, as seen in Figure 7.14.

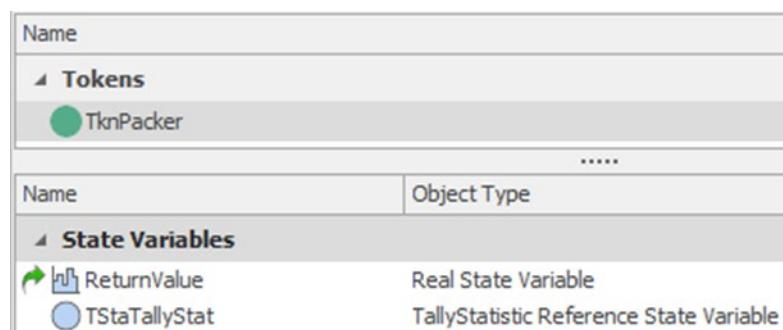


Figure 7.14: TknPacker State Variables

**Step 3:** Rename the **SrvPacker1\_Processing** add-on process to **SrvPacker\_Processing** and specify its properties, as seen in Figure 7.15. Change the type of TOKEN that will execute the process steps to

<sup>89</sup> Tokens always have the state variable *ReturnValue*.

**TknPacker.** Then specify the *Input Arguments* property with the one row (input arguments) as defined in Figure 7.16, which specifies the input parameter's name and how it is linked to the customized token.<sup>90</sup>

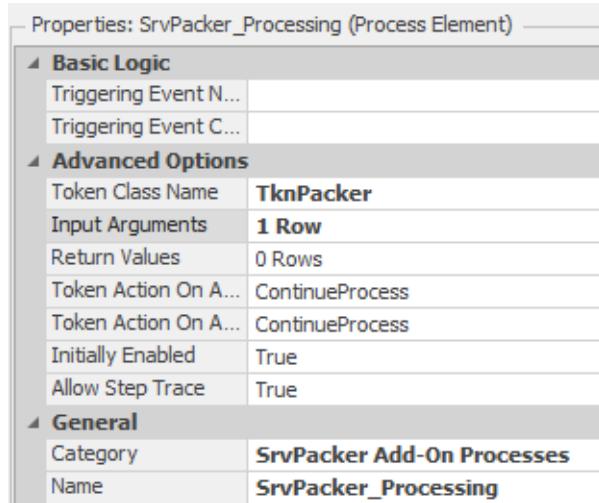


Figure 7.15: Properties of the SrvPacker Process

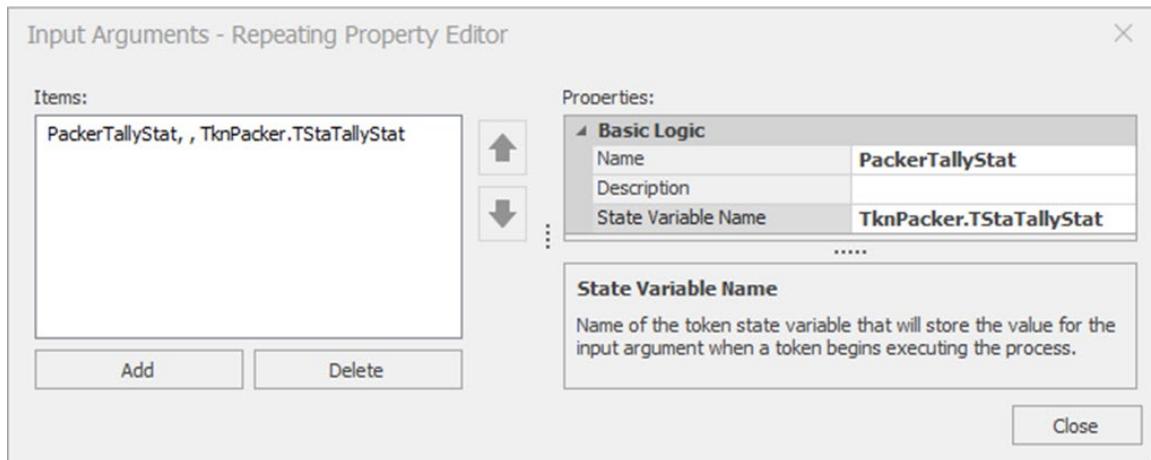


Figure 7.16: Input Arguments for the Process

**Step 4:** Next, in **SrvPacker\_Processing**, update the **TALLY step Tally Statistic Name** to **TknPacker.TStaTallyState** is shown in Figure 7.17.

---

<sup>90</sup> Notice, you can also specify the process return values which might be useful if one process calls another process.

Properties: TimeBeforePacked (Tally Step Instance)	
Basic Logic	
Tally Statistic Name	TknPacker.TStatTallyStat
Value Type	Expression
Value Expression	TimeNow - ModelEntity.TimeCreated

**Figure 7.17 Updated Properties of Tally Step**

**Step 5:** Finally, invoke the **SrvPacker\_Processing** at each packer by specifying its individual argument. Figure 7.18 shows the example for Packer 2.

Add-On Process Triggers	
Run Initialized	
Run Ending	
Entered	
Before Processing	
Processing	
Input Arguments	
Packer Tally Stat	TallyStatSystemTimePacker2

**Figure 7.18: Invoke the Tokenized Process**

**Step 6:** Execute the model and note the tally time in system statistics for each of the three packers.

## Part 7.7: Commentary

- SIMIO's process capability is one of its standout features. It allows for powerful and unique extension and customization. Using it requires a potential change in thinking about simulation modeling.
- A lot of different modeling can be done with resources, a feature we will explore more in later chapters.

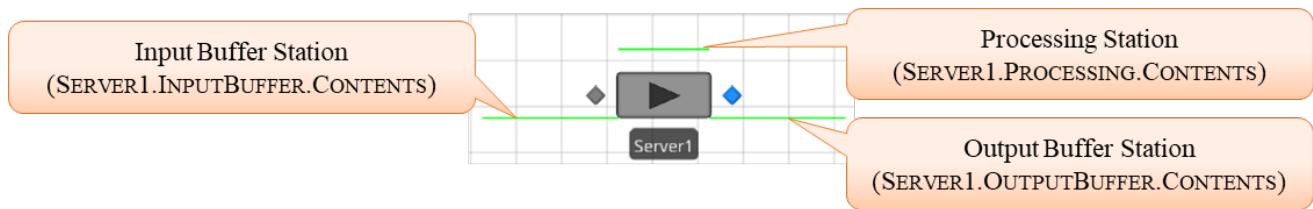
# Chapter 8

## Working with Flow and Capacity: The DMV

---

Almost every object in the SIMIO standard library has capacity, which means that their capacity may be seized and released, typically within the flow of entities - but the SERVER and the RESOURCE objects are used the most often. As seen in earlier examples, the SERVER is often the basic flow component of many models. When the RESOURCE has been employed, it is often a secondary resource to a SERVER. However, their relationship can be convoluted by the need to consider systems with multiple capacities in the context of complicated entity flow.

The SERVER object is composed of three stations (locations) where entities are held (i.e., INPUTBUFFER, PROCESSING, and OUTPUTBUFFER). The stations are animated with “contents” queues, each having its own capacity (see Figure 8.1). A BASICNODE is used as input to the SERVER, while a TRANSFERNODE is the output.



**Figure 8.1: The Server Object**

The capacity of the SERVER can be a fixed value or have its capacity controlled by a *WorkSchedule*. Recall that capacity refers to the number of entities that the object can simultaneously serve. When no capacity of the processing station is available and an INPUTBUFFER exists (i.e., Input Buffer capacity is greater than zero), then the entities wait in the INPUTBUFFER. The *Processing Time* property is the time required for the server to process an entity in the processing station and is a required property of the server.<sup>91</sup> The SERVER can occupy one of nine states, as explained in Figure 8.2 and SIMIO provides additional symbols to animate the server’s current state automatically. The state applies to all its capacity (i.e., if one or more units of its capacity are allocated, the server is considered processing), and statistics are collected on the amount of time spent in each state.

A RESOURCE is a much simpler object than the SERVER, but it shares a lot of common properties, including capacity specification, and is not composed of stations. Therefore, a RESOURCE has no place for entities to wait or be served, and no processing time specification is used to control when the resource is released. So, a resource must have its capacity seized and released externally in a SIMIO process, as seen previously.

Flow and capacity are often the primary concerns in many different types of systems (i.e., production, manufacturing, service, healthcare, etc.). Capacity may be used to restrict flow but cannot be arbitrarily expanded. So, finding ways to expand capacity and extend flow is a significant challenge, as simulation offers one way to experiment with different configurations of capacity and different flow mechanisms.

---

<sup>91</sup> The *Process Type* property can be changed to handle more complicate sequences rather just a “Specific Time” and will be explored in a later chapter.

Server State	Meaning	Additional Symbols
Starved (0)	Currently idle since no unit of capacity has been allocated.	 Starved (0)
Processing (1)	At least one unit of capacity has been allocated to process an entity.	 Processing (1)
Blocked (2)	The current entity has finished processing but cannot proceed into the network.	 Blocked (2)
Failed (3)	The Server has failed because of some condition.	 Failed (3)
OffShift (4)	Currently the capacity of the Server is zero meaning it is unavailable.	 OffShift (4)
FailedProcessing (5)	The Server failed while processing, but processing continues for entities in progress.	 FailedProcessing (5)
OffShiftProcessing (6)	The Server goes off shift while processing, but processing continues for entities in progress.	 OffShiftProcessing (6)
Setup (7)	At least one unit of capacity has been allocated and the Server is being setup for processing.	 Setup (7)
OffShiftSetup (8)	The Server goes off shift while being setup, but the setup continues for entities in progress.	 OffShiftSetup (8)

**Figure 8.2: Active Symbol Definitions**

## Part 8.1: The DMV Office

A Department of Motor Vehicles (DMV) often operates multiple offices at various sites across a state. If a simulation model can be created for one site, then perhaps it can be “reused,” with some minor changes, for other sites (and thus, the cost of model development is spread over several installations). A typical DMV office tests driver license applicants and supplies driver licenses to those passing the tests. A typical office size is being used as a reference for the analysis. The office typically opens at 8:30 am and closes at 4:30 pm each day to service license applicants. Applicants arrive more or less randomly, although the rates of arrival change during the day. There are three types of license applicants: Permit, New License, and Renewal. All arriving applicants go through registration, where one of the DMV clerks enters an applicant’s personal information into the computer and collects the license fee. Next, all applicants take a written and visual test on one of three test stations, each of which is staffed by its own DMV Tester. Not everyone passes these tests. Those getting a “New License” must also pass a driving test accompanied by a DMV officer, where some fail the driving test. People who fail any (written/visual or driving) of the tests leave the office and return another day. Those who pass all their tests have their picture taken and their license processed and printed. The DMV is interested in the office’s service to applicants and the use of DMV resources. Consequently, the flow times of applicants are a major concern. Since queuing times are a part of flow time, it is important. If congestion in the office becomes high, this also becomes a concern. Finally, since these are public facilities, the cost of the facilities, especially the use of DMV resources, must be balanced against the service to applicants.

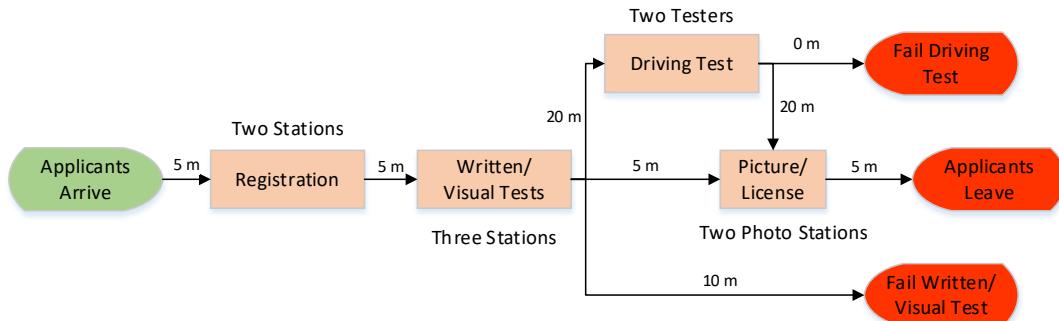
### Developing a Base Model

In any modeling activity, try to start with the simplest model you can develop quickly. Make whatever assumptions you need to get a model “up and running” right away. It’s alright if your assumptions don’t correspond to reality. We want something to work because we want to extend and evolve our model rather than create a final model at the beginning. Most people new to simulation modeling want their model to be close to the final one, even though it is their initial attempt. They discover that their model has all kinds of complexities that they have trouble unraveling when they encounter an error. Suppose they started with a

simple (maybe completely unrealistic) model that is working. In that case, they can embellish it incrementally and adapt it into a final working model with far less difficulty than trying to create the final model at the beginning. Finally, it is not possible to build a model that ultimately represents the real system in every detail. If that is your plan, then you are better off experimenting with the real system. Remember, a model is only an approximation of the real system. How detailed your model is depends on what you want the model to reveal. In general, your model needs only to be as detailed as needed to obtain the performance measures of interest. Any more detail is unnecessary.<sup>92</sup>

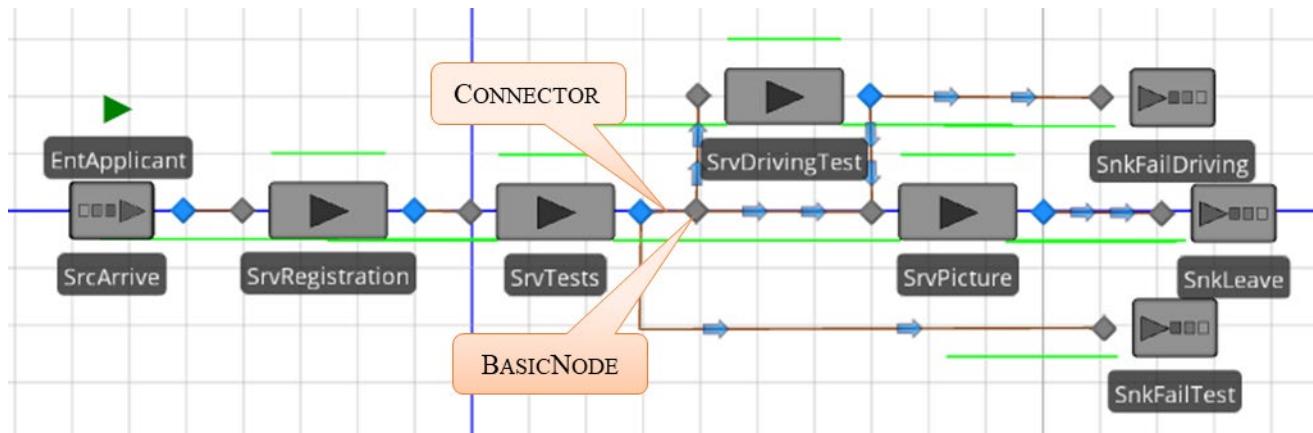
A second recommendation is not to worry about the input right away. The critical concern is getting a model that appears most closely to match the system so you can reliably obtain the needed performance measures. As you are building your model, it will become clear what input data is needed. Let the model inform your data collection so you don't spend a lot of time collecting unimportant or wrong data.

**Step 1:** Draw a flowchart of the service process, which can come from a “value-stream map”.



**Figure 8.3: DMV Flowchart**

**Step 2:** Since our flowchart in Figure 8.3 corresponds easily to the basic SIMIO objects, it can be used as a basis for the initial model. Use the following information to create a model with the appropriate objects as shown in Figure 8.4.



**Figure 8.4: Initial DMV Model**

- Insert a new MODELENTITY named **EntApplicants** with the *Initial Desired Speed* property set to “1” meter per second. The applicants arrive randomly but in a time-varying arrival process. Based on this

---

<sup>92</sup> A problem often arises when a client/boss sees the animated model not conforming exactly like the real system. Some people (naively) equate the “quality” of the model with the quality of the animation. Sometimes you are forced into animation details that have little to do with performance measures.

information, general estimates of the arrival rates are shown in Table 8.1. From the *Data* tab, insert a new RATE TABLE named **ArrivalRateTable** that has 15 intervals with an interval size of 30 minutes.

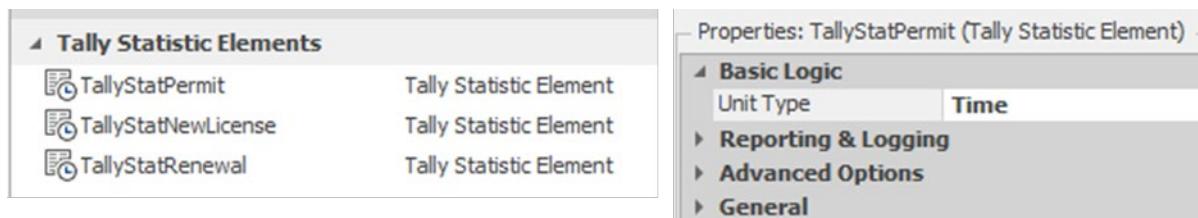
**Table 8.1: Hourly Arrival Rates for Applicants**

Time	Hourly Arrival Rate
8:30-9 am	15
9-9:30 am	10
9:30-10 am	8
10-10:30 am	12
10:30-11 am	15
11:00-11:30 am	18
11:30-noon	15
Noon-12:30 pm	12
12:30-1 pm	10
1-1:30 pm	8
1:30-2 pm	6
2-2:30 pm	7
2:30-3 pm	5
3:00 – 3:30 pm	5
3:30-4 pm	5

- Insert a new SOURCE named **SrcArrive** that uses the new arrival table **ArrivalRateTable**.
- The four basic activities will be modeled using SERVERS named **SrvRegistration**, **SrvTests**, **SrvDrivingTest**, and **SrvPicture**, as seen in Figure 8.4.
- From Figure 8.4, notice the BASICNODE named **BNodePass** that was inserted after **SrvTests**, which is not part of the flow chart but will be used to help branch applicants who fail the written or eye test. Use a CONNECTOR between the **SrvTests** SERVER and the BASICNODE so it doesn't take any time. Use a 20-meter and a five-meter path between the BasicNode and SrvDrivingTest and SrvPicture, respectively. Connect all other objects via paths based on the distances from the flow chart.
- Finally, insert three SINKS named **SnkLeave**, **SnkFailTest**, and **SnkFailDriving**.<sup>93</sup>

**Step 3: Recall there are three types of applicants (i.e., permits, new licenses, and renewal licenses).**

- To keep track of time in the system for each type, insert a TALLY Statistic, as seen in Figure 8.5. Make sure your TALLY statistics have **Time** as the *Unit Type*.



**Figure 8.5: Creating Tally Statistics to Track Time in the System for Each Type of Applicant**

- Insert a new data table from the Data tab named **TableApplicant** with six different properties with the following particular characteristics defined in Table 8.2 and shown in Figure 8.6.

<sup>93</sup> Note, different SINKS will be used to separate the statistics for those that fail test versus not. From an animation standpoint, one can put the three SINKS on top of one another to demonstrate the applicants leaving the same door so it visually looks correct.

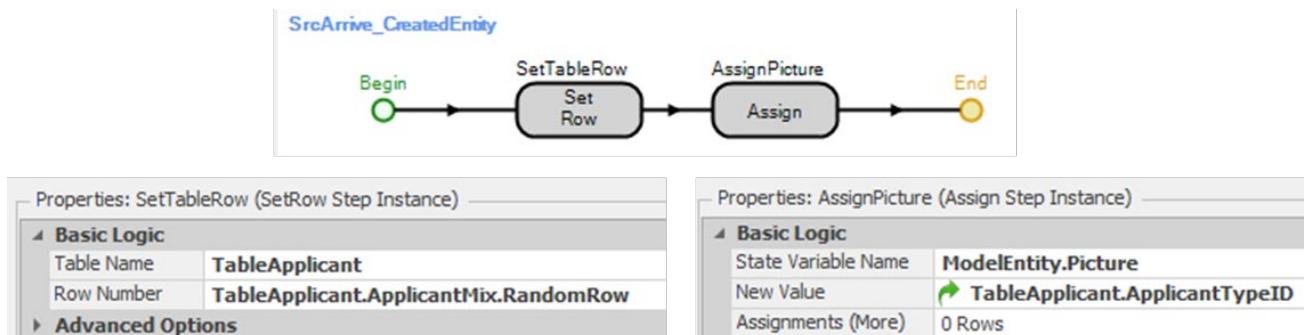
**Table 8.2: Applicant Characteristics Data Table**

Property Name	Property Type	
Name	String	The name of the applicant type
ApplicantType	Integer	It will be used to set the entity's animation picture
ApplicantMix	Real	The percentage of time this applicant type arrives
RegistrationTime	Expression ( <i>Unit Type</i> set to “Time” with minutes as the <i>Default Units</i> ).	The registration time varies by the applicant type.
PassTests	Real	The probability of passing varies by applicant type
PassDrivingTest	Real	Probability of passing the driving test
TallyStatTimeInSystem	Tally Statistics Element	Used to track each type’s time in system

Table Applicant							
	Type	Applicant Type ID	Applicant Mix	RegistrationTime (Minutes)	Pass Tests	Pass Driving Test	Tally Stat Time In System
1	Permit	0	25	Random.Pert(6,8,10)	90	0	TallyStatPermit
2	NewLicense	1	25	Random.Pert(8,10,14)	95	90	TallyStatNewLicense
3	Renewal	2	50	Random.Pert(5,7,8)	98	0	TallyStatRenewal

**Figure 8.6: Table Values for the Three Applicant Types.**

- Select the **EntApplicant** and add two additional symbols, coloring them red and blue, respectively.<sup>94</sup>
- Next, as **EntApplicants** are created by the **SrcArrive**, assign the applicant type and associated picture using the *Created Entity Add-on Process Trigger*, as seen in Figure 8.7.<sup>95</sup>

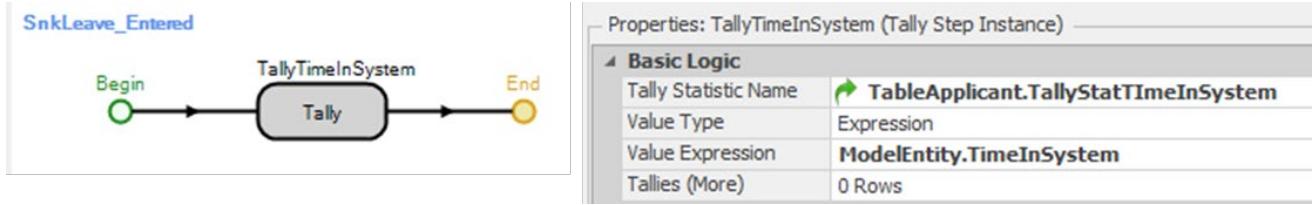


**Figure 8.7: Assigning the Applicant Type and Picture**

- Once applicants leave the system, tally the time in the system through the *Entered Add-on Process Trigger* of the **SnkLeave**, as shown in Figure 8.8.

<sup>94</sup> Switch to 3-D mode to color the entire entity widget.

<sup>95</sup> We have a strong preference for processes over the specifications in objects (e.g., the table, state, tally properties), since processes give use the most flexibility in specifying and re-specifying these additions to the model.



**Figure 8.8: Tally Time in System**

*Question 1:* Will these Tally statistics by applicant type include the time in system for those applicants who fail any of the tests (why or why not)?

---

**Step 4:** Use Table 8.3 to set the processing times and capacities for each of the four servers.

**Table 8.3: Processing Time Property Values**

SERVER	Processing Time Property Value	Initial Capacity
SrvRegistration	TableApplicant.RegistrationTime	2
SrvTests	Pert(10, 14, 20) minutes	3
SrvDrivingTest	Pert(15, 25, 40) minutes	2
SrvPicture	Pert(1, 2, 3) minutes	2

*Question 2:* Why would the Pert distribution be preferred over the Triangular for *Processing Time*?

---

**Step 5:** If you save and run the model, the applicants will choose the various branches equally likely since the *Selection Weight* properties are all set to one. Recall that only new licenses go through the driving test, and the passing rate of the driving and written/eye tests are stored in the table. Change the *Selection Weight* properties for the paths so they account for how the applicant types should branch, as seen in Table 8.4.

**Table 8.4: Branching Conditions**

From Node	To Node	Selection Weight Property
Output@SrvTests	BASICNODE named BNodePass	TableApplicant.PassTests
Output@SrvTests	Input@SnkFailTests	100 - TableApplicant.PassTests
BNodePass	Input@SrvDrivingTest	TableApplicant.ApplicantTypeID == 1
BNodePass	Input@SrvPicture	TableApplicant.ApplicantTypeID != 1
Output@SrvDrivingTest	Input@SrvPicture	TableApplicant.PassDrivingTest
Output@SrvDrivingTest	Input@SnkFailDriving	100 - TableApplicant.PassDrivingTest

**Step 6:** Save and run the model, observing it for a while before fast-forwarding to complete one replication.

*Question 3:* What are the average times in system for New License, Permit, and Renewal applicants?

---

*Question 4:* What are the stations' observed "ScheduledUtilization" for Driving Tests, taking Pictures, Registration, and Tests?

---

*Question 5:* Do you have any concerns relative to the model's behavior or the results?

---

*Question 6:* Why do we need to be careful about drawing conclusions from a simulation composed of only one replication?

---

*Question 7:* How many units of capacity need to be unavailable for the resource to occupy the “Offshift(4)” state?

---

*Question 8:* What bothers you about the model realism, as seen in its animation, based on your experience in a DMV office?

---

## Part 8.2: Using Resources with Servers

After building the initial model, it was discovered that the people who register applicants also take pictures and hand out the licenses. These DMV clerks work in the same physical area and share the workload, meaning there are four clerks to perform both functions. However, when we ask about the working environment, we find that only three applicants can be processed at once at the registration and that only two people can have their picture taken at the same time.

**Step 1:** Change the capacity of the **SrvRegistration** to three, meaning that no more than three entities can be processed at the same time on this server. The **SrvPicture**'s capacity will remain two.

**Step 2:** Insert a **RESOURCE** object into the model named **ResClerks** with an initial capacity of four, which corresponds to the four clerks. A resource object can have its capacity “used” at several servers within a model, while server capacity is limited to a specific server.

*Question 9:* Even though there are four clerks, why can only three be used during the registration process at the same time?

---

**Step 3:** Next, under the *Secondary Resources*<sup>96</sup> section at both the **SrvRegistration** and **SrvPicture** servers, use the “*Resource for Processing*” property and specify the use of **ResClerks**<sup>97</sup>, as seen in Figure 8.9.

Secondary Resources	
For Processing	
Repeat Group	<b>False</b>
Resource Type	Specific
Resource Name	<b>ResClerks</b>
Selection Goal	Preferred Order
Request Move	None

**Figure 8.9: Specifying the use of Clerks at the Server**

**Step 4:** Save and run the model.

---

<sup>96</sup> It is convenient to use the *Secondary Resources* when you don't need the flexibility of the add-on processes. Resources for Processing does both the seizing and releasing of the resources.

<sup>97</sup> By default the capacity requested will only be one. You have more flexibility with the *Seize* step.

*Question 10:* Does the model now behave as expected?

---

*Question 11:* Did you notice that the RESOURCE shows its “Busy (1)” state whenever any of its capacity is being used, similar to the Processing state of the server?

---

If a server employs resources for service, then the capacity to serve entities may be limited by either the resource capacity availability, the server capacity availability, or both. If only resources limit the capacity, then the server capacity should be made arbitrarily large (i.e., infinity) so as to not constrain the system.

### Part 8.3: Handling Failures

Suppose we now discover that the reception area is interrupted by phone calls. These phone calls arrive randomly (i.e., Exponentially distributed) on average every 30 minutes, and they take anywhere from 1 to 4 minutes, with 2 minutes being typical.

Both SERVERS and RESOURCES can experience failures, as can WORKSTATIONS, COMBINERS, SEPARATORS, VEHICLES, and WORKERS. The failure options are found under the object’s “Reliability Logic” property section. In SIMIO, the failure concept is intended to be used for any downtime, exception, or (non-modeled) interruption<sup>98</sup>. Four different types of failures can be specified, two of which are based on “counts” while the others are based on time, as described in Table 8.5.

**Table 8.5: Different Types of Failure Types**

Failure Type	Description
No Failures	No failure or exception will occur.
Calendar Time Based	The user specifies the time between exceptions or failures. Often used model general reliability or maintenance programs.
Event Count Based	The user specifies that an exception or failure will occur after a particular number of events has happened (e.g., the number of tool changes or a specific number of arrivals).
Processing Count Based	The user specifies that an exception or failure will occur after a certain number of operations have occurred on the SERVER.
Processing Time Based	The user specifies that an exception or failure will occur after the object has operated a certain amount of time (e.g., after 100 hours of operation the machine has to go through maintenance).

When an object is interrupted, its entire capacity is interrupted – its state is changed to “Failed(3)”. So, do we interrupt the service at the server, or do we interrupt the resource?

**Step 1:** One can use a “*Calendar Time Based*” type failure to model the phone call interruption. Select the **SrvRegistration** SERVER object and specify the “reliability logic,” as shown in Figure 8.10, using the *Uptime Between Failure* and the *Time to Repair* expressions corresponding to the phone calls.

---

<sup>98</sup> An *Interrupt* step can be used to model interrupts.

Reliability Logic	
Failure Type	Calendar Time Based
Uptime Between Failures	Random.Exponential(30)
Units	Minutes
Time To Repair	Random.Pert(1,2,4)
Units	Minutes

**Figure 8.10: Specifying the Failure**

**Step 2:** Implement this reliability logic in the **SrvRegistration**. Run the model and observe its behavior.

*Question 12:* Did you see the interruptions displayed by the server being colored red?

---

*Question 13:* If you simulate in fast-forward, the simulation ends at 7.5 hours; how many phone calls were handled at the registration desk, and what percent of the time did the registration desk fail?

---

*Question 14:* Did the failures of the server cause the secondary resource to fail? What was its state while the server failed?

---

*Question 15:* Did the interruptions in the server affect the “TimeProcessing” in the registration?

---

**Step 3:** In reality, the clerks are the ones that need to answer the phone calls. Take the failure off the server and put it on the resource **ResClerks**. Run the model and observe its behavior.

*Question 16:* Did you see the interruptions displayed by the resource being colored red?

---

**Step 4:** If you simulate in fast-forward, the simulation ends in 7.5 hours. Examine the resource statistics for **ResClerks**, and you will notice “TimeFailed” and “TimeFailedBusy” state statistics. The “Failed(3)” state occurs when the resource is not busy, and the “FailedBusy(5)” state occurs when it is busy (at least one unit of capacity is being used). Notice that the colors for both states are the same, so you can’t tell between the two failed states in the animation.

*Question 17:* From the statistics on the resource, how many phone calls were handled by the resource when it was idle, and how many phone calls were handled when the resource was busy?

---

*Question 18:* Did the interruptions in the resource affect the “TimeProcessing” in the registration?

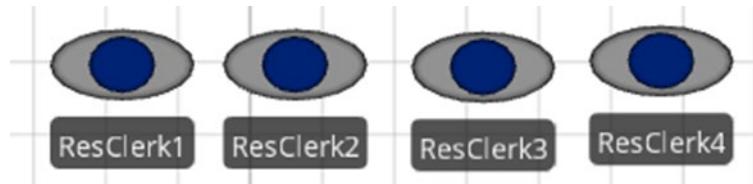
---

*Question 19:* Why do the interruptions of resources cause an increase in the time being processed?

---

**Step 5:** Unfortunately, failing the entire server or resource is probably not our best option for modeling the handling of phone calls. Since the phone calls are answered individually, we need to spread them over the clerks. One possibility would be to create entities to represent the phone calls and have them interrupt existing service. However, let’s use the pre-defined reliability logic for our present purposes. However, each clerk needs to handle phone calls individually, so we need to have individual resources. Replace our **ResClerks** with four separate **RESOURCE** objects, each with capacity one (see Figure 8.11). Since phone

calls occur about 30 minutes apart (on average), we will specify the time between phone calls to be 120 minutes for each clerk, using the previous time to repair.<sup>99</sup>



**Figure 8.11: Individual Clerks**

**Step 6:** Unfortunately, secondary resources<sup>100</sup> can only seize a single specific object (i.e., the **ResClerks**, which had a capacity of four), but now there are four separate RESOURCES that need to be seized. From the “Definitions” tab, create an OBJECT LIST called **ListClerks** that contains the four clerk Resources, as seen in Figure 8.12. The list will allow us to select one of the clerks based on some criteria.

A screenshot of the SIMIO software's 'Objects' tree view. Under 'ListClerks', there is a table with a single column labeled 'Object'. The table contains four rows, each labeled 'ResClerk1', 'ResClerk2', 'ResClerk3', and 'ResClerk4' respectively.

**Figure 8.12: Creating an Object List of Resources**

**Step 7:** At both the **SrvRegistration** and **SrvPicture** SERVERS, under the *Secondary Resources* section for *Resource for Processing*, choose the “Select From List” option as the *Object Type* property. Specify the **ListClerks** as the *Object List Name* property value, as seen in Figure 8.13. Use “Random” as the *Selection Goal*, which forces SIMIO to use the clerks at the servers uniformly.

A screenshot of the SIMIO software's configuration interface for 'Secondary Resources'. Under 'For Processing', the 'Resource Type' is set to 'Select From List', 'Resource List Name' is set to 'ListClerks', and 'Selection Goal' is set to 'Random'. Other settings include 'Repeat Group' (False) and 'Request Move' (None).

**Figure 8.13: Select a Clerk**

**Step 8:** Save and run the model, look at the animation, and finally, look at the statistics at the end of 7.5 hours. Now, four resources are being modeled, so there are individual statistics for each.

*Question 20:* What is the scheduled utilization for each of the clerks?

---

*Question 21:* How many phone calls did each of the clerks process while they were idle?

---

<sup>99</sup> You can modify the existing resource object so it has capacity one and its *Uptime Between Failures* as Random.Exponential(120) minutes. Then copy the object three times and name them accordingly.

<sup>100</sup> The *Seize* process step has the same issue of being able to seize from a single object.

Question 22: How many phone calls did each clerk process while they were busy?

---

## Part 8.4: Server Configuration Alternatives

Sometimes, there are desirable alternative configurations for the servers in a model, often motivated by animation needs or handling separate failures. For example, the **SrvTests** object in the current model represents the processing of up to three applicants at the testing stations. Perhaps it would be better to represent this service as three separate SERVERS. Also, the evaluators that administer the test need to take a break after processing between eight or twelve applicants, so separate SERVERS will be needed.

**Step 1:** The evaluators that assess the written and eye tests take a break after processing between eight and twelve applicants. The reliability logic can be used to handle this type of break. Set the *Failure Type* property to “Processing Count Based” to count the number of applicants where *the Count Between Failures* set to a uniform distribution between eight and twelve with the time of the break taking between two and six minutes, but generally they take four minutes as seen in Figure 8.14.

Reliability Logic	
Failure Type	Processing Count Based
Count Between Failures	Random.Discrete( 8,29,4,10,6,11,8,12,1,0 )
Time To Repair	Random.Pert(2,4,6)
Units	Minutes

Figure 8.14: Taking a Break Based on the Number of Applicants Processed

**Step 2:** Since the evaluators doing the test will not break at the same time, we need to have individual test stations. Change the name of **SrvTests** to **SrvTests2** and the capacity back to “1.” Copy it two times, creating a total of three SERVER objects, as shown in Figure 8.15. Connect the test stations with the registration using five-meter PATHS. We will assume that the branching to each testing station is equally likely (i.e., equal *Selection Weights*). Connect the other two tests with CONNECTORS to the **Ouput@SrvTests2**, which allows us to use the same branching logic to drive test, picture taking, or failure.

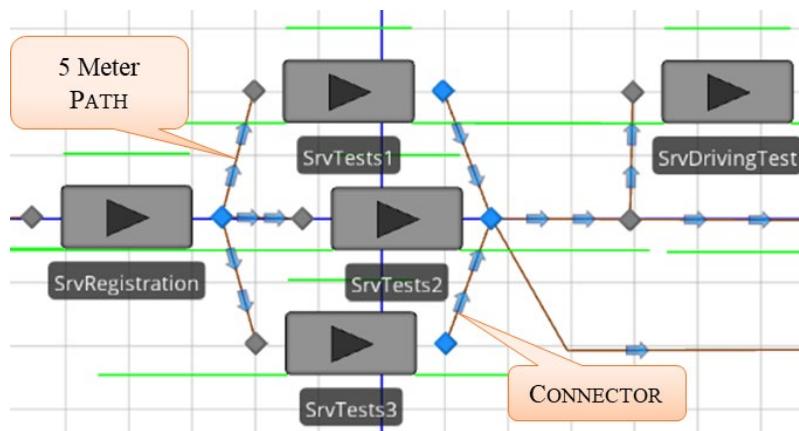


Figure 8.15: Separate Testing Stations

**Step 3:** Now save and run this model while observing the animation.

Question 23: How do the applicants choose which of the three testing stations to enter?

---

Question 24: Is there anything about the behavior of this model that causes you concern?

---

**Step 4:** Typically, there is one waiting station after registration where they wait for the next available testing station. To prevent applicants from waiting at the testing stations, first set the input buffer capacity of each testing server to zero under the *Buffer Capacities* section of the SERVER object.

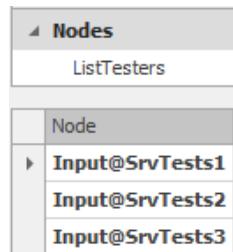
**Step 5:** Save and rerun the model, observing the animation.

*Question 25:* Now, how do the applicants wait at the testing stations? Is that a desired behavior?

---

The applicants now wait at the input node of the testing stations and are unable to enter the SERVER's station since the input buffer capacity is zero. Furthermore, the entities seem to make a decision about which path to take independent of the traffic on that path, the number at the associated test station, or if the tester is on break.

**Step 6:** To handle those issues, the branching process needs to be modified to choose an appropriate testing station. From the “*Definitions*” tab, create a NODE LIST called **ListTesters** that contains the input node (i.e., **Input@**) for each of the three testers, as seen in Figure 8.16. A list will allow SIMIO to select one of the input nodes using some criterion.



**Figure 8.16: Creating a Node List of Testers**

**Step 7:** Select the **Output@SrvRegistration** transfer node that connects the registration to the servers, change the *Entity Destination Type* property to “Select From List,” and select the **ListTesters** list that was just created in the *Node List Name* property.

- Set the *Selection Goal* property to be the **Smallest Value** with the *Selection Expression*<sup>101</sup> equal to `Candidate.Node.AssociatedStationload`<sup>102</sup>, which chooses the server with the shortest line by determining the number that is routed to the SERVER plus the number in the Contents Queues of the InputBuffer<sup>103</sup> and Processing<sup>104</sup> stations, as seen in Figure 8.17.
- If a tester is taking a break, applicants should not be routed to this person while they are on the break. To avoid selecting these nodes, the *Select Condition* property can be used to include only certain items from a list. This expression property has to evaluate to “True” before the CANDIDATE NODE object will be considered for selection for routing. Therefore, we will set the property to select only those servers that have not failed before choosing the smallest line.

---

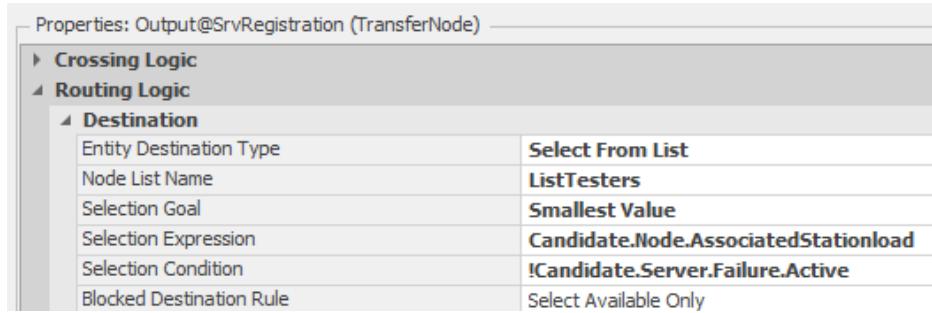
<sup>101</sup> Do not confuse the *Selection Expression* property with the *Selection Condition* which is used to include/filter candidate nodes.

<sup>102</sup> Note the CANDIDATE object has to be used to specify a wildcard like “\*”. SIMIO will replace the CANDIDATE object with the appropriate object from the list when determining the smallest value. Also, if there is a tie between nodes they will be selected in the order they were placed into the list.

<sup>103</sup> The Contents queue of the InputBuffer Station is used to hold entities waiting to be processed by the SERVER.

<sup>104</sup> The Contents queue of the Processing Station represents the location of the entities that are currently being processed by the SERVER.

- The `Failure.Active` function will be “False” if the server is currently not in a failure mode. If the server has not failed, then allow it to be selected as a candidate using the expression `!Candidate.Server.Failure.Active`, as seen in Figure 8.17.<sup>105</sup>



**Figure 8.17: Choosing the Shortest Line**

**Step 8:** Save and run the model, observing what happens to the entities that leave registration. You may need to increase the time of the breaks to make sure the selection condition is working.

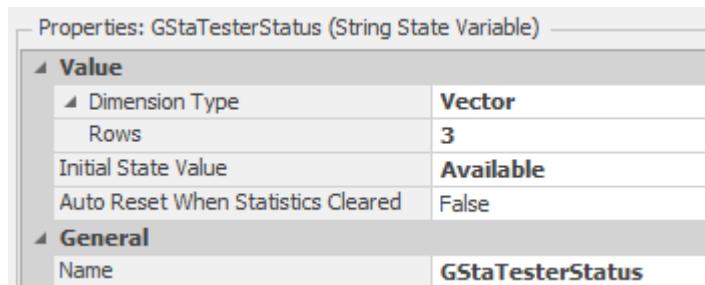
*Question 26:* Now, does the model appear to be behaving as you want?

---

## Part 8.5: Restricting Entity to Flow to Parallel Servers

Now, applicants are forced to wait in the registration output node until a tester is available (i.e., the SERVER is empty) because the traveler capacity paths are one, and the servers' input buffer are zero. Chapter 14 will present a more complete explanation of various blocking methods. Sometimes, it is necessary to block flow under somewhat complex circumstances. Suppose that the testers who give the visual and written exam must enter the exam results into a computer. The applicant is allowed to return to the waiting area, but the tester remains busy to complete the process. Obviously, the tester is unavailable to other applicants until the results are entered.

**Step 1:** To keep track of the status of the tester, we will utilize a STATE VARIABLE associated with each tester. From the *Definitions→States* section, insert a new discrete String state variable named **GStaTesterStatus**. Since there are three rooms, make the *Dimension Type* property a **Vector** and set the number of *Rows* to three with an *Initial State Value* of Available<sup>106</sup> as seen in Figure 8.18.<sup>107</sup>



**Figure 8.18: Setting a Vector State Variable**

**Step 2:** From the *Animation* tab, insert three status labels beside each of the tester SERVERS with an expression associated with the **GStaTesterStatus** (e.g., `GStaTesterStatus[1]` for the first room as

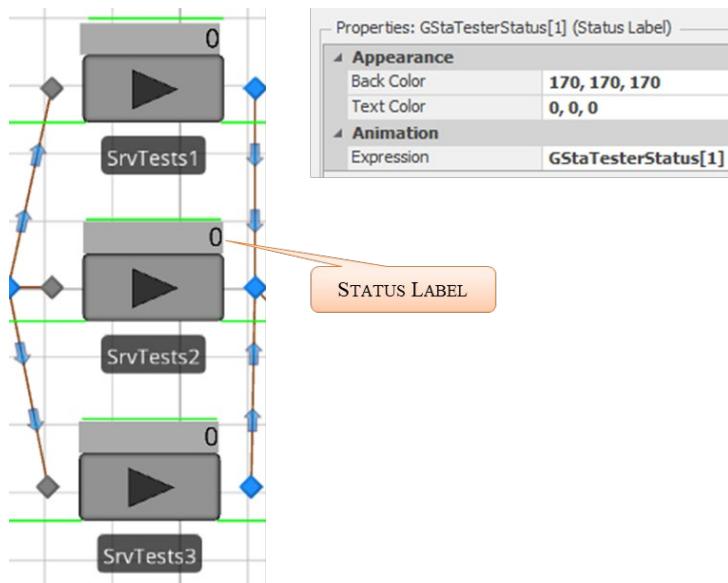
---

<sup>105</sup> The `!` or Not operator can be used to take the complement of a variable (i.e., `! False` would equal True).

<sup>106</sup> Here “Available” is a string constant (arbitrarily chosen), not a numerical value.

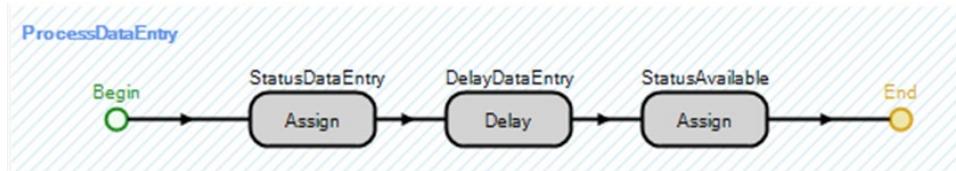
<sup>107</sup> Note we could have created three individual state variables (i.e., `GStaTesterStatus1`, `GStaTesterStatus2`, and `GStaTesterStatus3`) instead of the one variable that is a vector of size three.

seen in Figure 8.19). Specify all three expressions accordingly. Vectors start with an index of one, and each row is specified in brackets.



**Figure 8.19: Adding Status Labels**

**Step 3:** Once an applicant has been seen by one of the testers, we will need to set the status of the tester to “DataEntry,” delay for five to ten minutes, and then set the status to “Available.” We will model this via a specialized process for each room. Navigate to the “Processes” Tab from the *Process* section, and use the *Create Process* button to create a new process named **ProcessDataEntry**, as seen in Figure 8.20. Note that this is a case where *State Assignments* cannot be used to model the scenario.



**Figure 8.20: Process for Data Entry and Changing the Status of the Tester**

- Insert an *Assign* step that will set the first room’s status (i.e., `GStaTesterStatus[1]`) to “DataEntry”.
- Insert a *Delay* step with a Delay Time expression set to `Random.Uniform(3, 6)`. For a moment, we will leave the Units set to hours, which allows us to see any potential problems.
- Copy the first *Assign* step and change the *Value* to “Available.”

Properties: StatusDataEntry (Assign Step Instance)		Properties: DelayDataEntry (Delay Step Instance)		Properties: StatusAvailable (Assign Step Instance)	
Basic Logic		Basic Logic		Basic Logic	
State Variable Name	GStaTesterStatus	Delay Time	Random.Uniform(3,6)	State Variable Name	GStaTesterStatus
Row	1	Units	Minutes	Row	1
New Value	"DataEntry"			New Value	"Available"

**Figure 8.21: Process Steps and Values for the ProcessDataEntry**

**Step 4:** The new process needs to be invoked after the applicant has finished processing. In the Facility window, select **SrvTests1** and specify the **ProcessDataEntry1** as the *After Processing* add-on process trigger which is invoked immediately after the entity has finished processing.

Add-On Process Triggers	
Run Initialized	
Run Ending	
Entered	
Before Processing	
Processing	
After Processing	ProcessDataEntry
Exited	

Figure 8.22: Using the ProcessDataEntry as the *After Processing* Add-On Process Trigger

**Step 5:** Save and run the model.

**Question 27:** What happens to the applicant who finishes service and then causes the data entry to happen?

---

**Step 6:** You should have noticed that the entity does not leave the tester until after the data entry is complete, which is not what we wanted. The reason this occurs is that the MODEL ENTITY is delayed, and not just the tester. The *Exited* add-on process trigger is invoked after the entity has physically left the SERVER. Instead of using the *After Processing*, specify the **ProcessDataEntry** process as the *Exited* add-on process trigger.

**Step 7:** Save and run the model.

**Question 28:** What happens to the applicant who finishes service now and causes the data entry to happen?

---

**Question 29:** What happens to the next applicant waiting for this tester?

---

In the first case, the current entity was delayed and did not leave the tester. When using the *Exited* trigger, the current applicant leaves the room, but the next applicant is not blocked, will enter the tester and begin processing. Also, the tester's utilization is incorrect because they are idle when performing the data entry.

**Step 8:** Therefore, we need to seize the tester to prevent it from being freed when the applicant leaves the room and release the server once the data entry is finished. Insert a *Seize* and *Release* steps to the **ProcessDataEntry** process as shown in Figure 8.23.

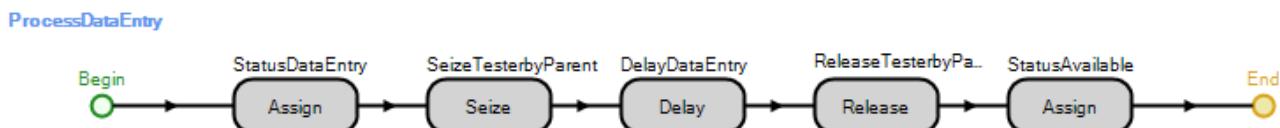
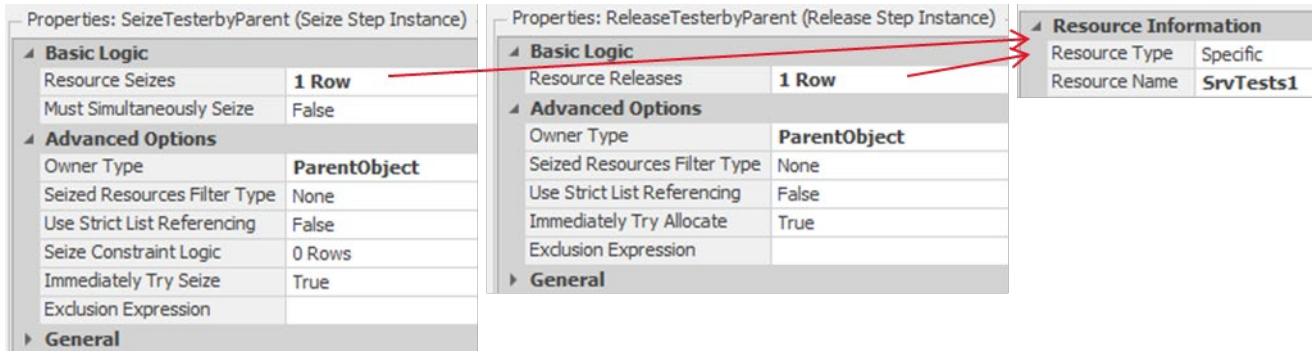


Figure 8.23: Adding a Seize and Release of Tester1

- Because the entity has left the server, it cannot be used to seize the tester. Therefore, we will use the *ParentObject*, which is the *MODEL* in this scenario, as the *Owner Type*, for both the *Seize* and *Release* process steps, as seen in Figure 8.24.



**Figure 8.24: Seize and Release SrvTest1 by ParentObject**

**Step 9:** Save and run the model, observing whether the **SrvTest1** server does not accept entities now.

**Question 30:** Does the SERVER accept entities when performing the data entry?

---

**Step 10:** The entities are still allowed to enter the path to the server even though the server is busy. Therefore, we need to block the server or the path using a different mechanism.<sup>108</sup> We need to edit the *Routing Logic* for the **Output@Registration TRANSFERNODE**. Recall the *Selection Condition* property of the *Routing Logic* section, which must evaluate as “True” in order for that particular node to be selected. Currently, the condition will not select testers that have failed. Therefore, modify the condition to include not choosing servers that are currently processing (i.e., resource state equal to one), as seen in Figure 8.25.

`!Candidate.Server.Failure.Active && Candidate.Server.ResourceState != 1109`



**Figure 8.25: Restrict Testers that are Busy from Being Selected**

**Step 11:** Save and run to make sure that the SERVER does not accept entities now.

**Question 31:** So, if the server’s resource state is one (i.e., “Processing”) when doing data entry, how does this expression ensure that only idle servers are eligible for selection?

---

**Question 32:** Does the SERVER **SrvTests1** accept entities while doing data entry now?

---

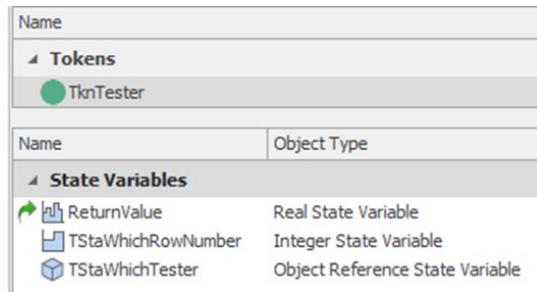
**Step 12:** Since the model is working properly now, change the *Units* of time to “Minutes” for the *Delay* step of **ProcessDataEntry**.

<sup>108</sup> One could shut down the path by changing the capacity or the direction to None or potentially cause the TransferNode Current Travel Capacity state variable to go to zero.

<sup>109</sup> Note, the && is for logical “And” such that we only select servers that have not failed and currently not processing someone.

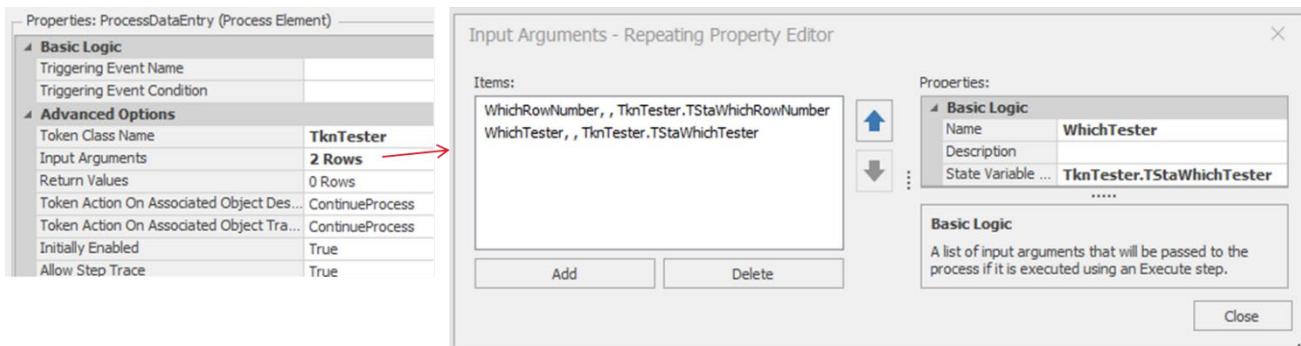
**Step 13:** At this point, we could select the **ProcessDataEntry** process and copy it two times. Then change the names of the two copied processes to **ProcessDataEntry2** and **ProcessDataEntry3**, the two **Assign** steps (i.e., the **Row** property set to two or three), the **Seize** step, and the **Release** step to specify the correct tester. However, to do this more generally (i.e., have the same process used by all three testers), we will use a “Tokenized Process.” This approach will be especially useful if additional testing stations are added later.

- From the *Definitions* tab, define a new TOKEN named **TknTester**. Add an INTEGER STATE variable named **TStaWhichRowNumber**, and from the *Object Reference* section, add an OBJECT REFERENCE STATE variable named **TStaWhichTester**, as seen in Figure 8.26.



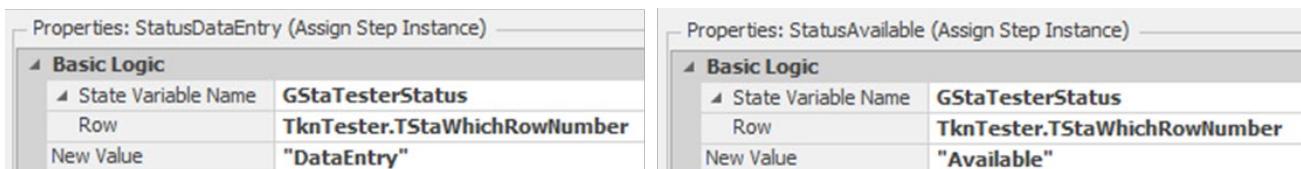
**Figure 8.26: New Token TnkTester to be used in a Tokenized Process**

- Edit the **ProcessDataEntry** process. Modify the process’s *Advanced Options* properties by specifying the process that will be executed by the **TknTester** TOKEN, as shown in Figure 8.27. Next, specify the process will take two input arguments (i.e., **WhichRowNumber** linked to **TknTester.TStaWhichRowNumber** and **WhichTester** linked to **TknTester.TStaWhichTester**).



**Figure 8.27: Adding Token Information to the Process**

**Step 14:** Next, change the two **Assign** steps within the process to employ the **TStaWhichRowNumber** as the **Row** property to update the **GStaTesterStatus** passed to the TOKEN **TknTester**, as seen in Figure 8.28.



**Figure 8.28: Use the Token Argument**

**Step 15:** Update the **Seize** and **Release** steps to use **TStaWhichTester** as the specific *Object Name*.

Resource Information	
Resource Type	Specific
Resource Name	TknTester.TStaWhichTester
Selection Goal	PreferredOrder

Figure 8.29: Using the TknTester to Specify Which Server to Seize and Release

**Step 16:** From the facility window, select **SrvTests1** and set the *Exited* add-on process trigger, as seen in Figure 8.33, to pass in the correct row number and tester. Repeat this for the other two testers, setting the row number and which tester to the correct values (i.e., “2” and “3” and **SrvTests2** and **SrvTests3**, respectively).



Figure 8.30: Invoking the Exited Process with its Argument

**Step 17:** The status labels currently change from “Available” to “Data Entry” but remain “Available” while the SERVER is processing an applicant. We need to update the status to busy when the tester is servicing an applicant to fix this issue.

- Since it will be similar for all three testers, add another “Tokenized Process” named **ChangeStatustoBusy**, as seen in Figure 8.31. Next, use the Token **TknTester** as the *Token Class Name* property and insert one input argument (i.e., the row number).

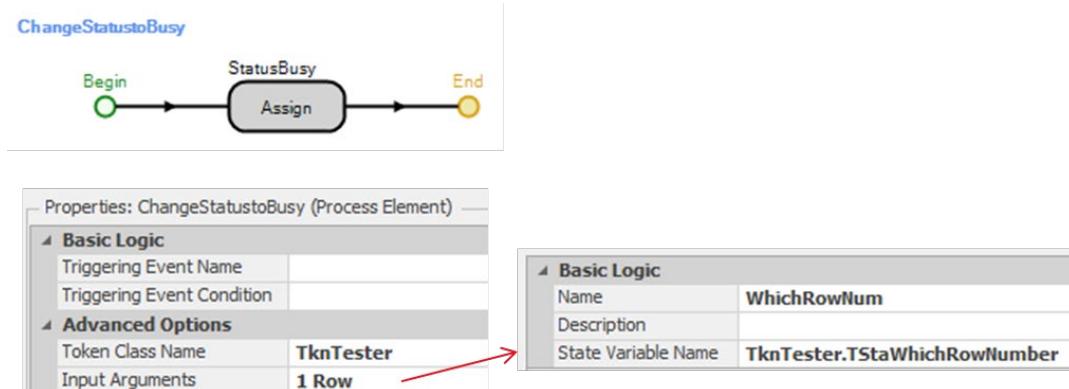


Figure 8.31: Creating the Process that will update the Status to Busy

- Insert an **Assign** step that will set the **GStaTesterStatus** variable to “Busy,” as seen in Figure 8.32.

Properties: StatusBusy (Assign Step Instance)	
<b>Basic Logic</b>	
State Variable Name	<b>GStaTesterStatus</b>
Row	TknTester.TStaWhichRowNumber
New Value	“Busy”

Figure 8.32: Setting the Status to Busy in the **Assign** Step

**Step 18:** For each of the three tester SERVERS, specify the *Before Processing* Add-on Process triggers to use the **ChangeStatustoBusy** process specifying the correct row number, as seen in Figure 8.33 for the first tester.

Before Processing	ChangeStatusToBusy
Input Arguments	
Which Row Num	1

**Figure 8.33: Changing the Room Status to Busy**

**Step 19:** Save and run the model, ensuring the status labels turn to “Busy,” and the model works correctly.

**Question 33:** What is the average utilization of each server?

---

**Question 34:** What is the total time an applicant has to wait for a tester?

---

**Question 35:** What is the total cycle time for each license type?

---

**Question 36:** Are there any drawbacks to this new model?

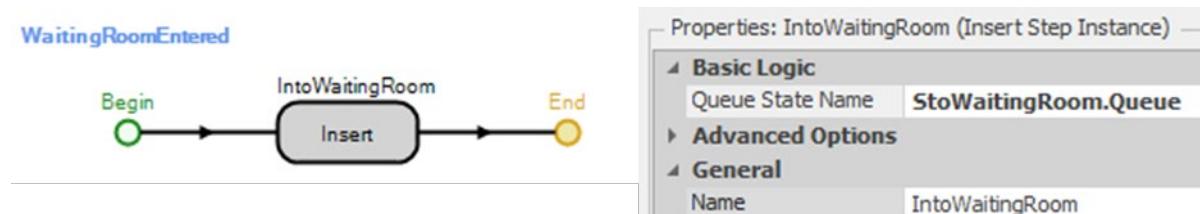
---

## Part 8.6: The Waiting Room Size

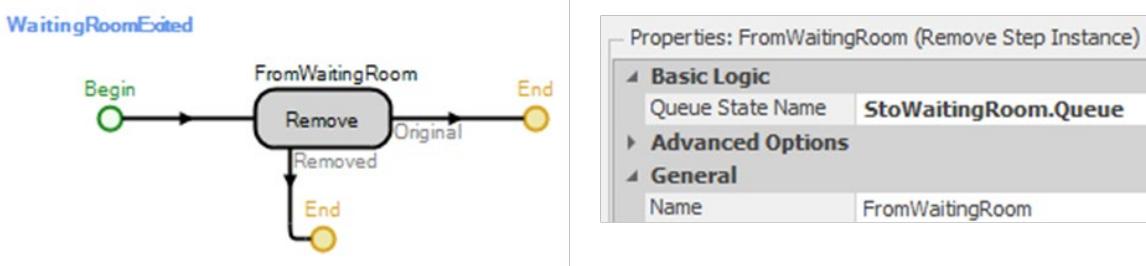
In the real system, people wait for services in a common waiting area, except for those waiting for registration, who queue in front of the registration. This common area consists of waiting for the tester, the driving test, and the picture/license. Looking at each waiting area separately ignores the fact that contributions to the waiting area change throughout the day. To visually see what is happening in the common waiting area, we need to introduce the SIMIO STORAGE object, which will be used to represent all the waiting areas together. (Storages do not actually contain entities, but entities hold membership in storages.)

**Step 1:** From the *Definitions→Elements* section, insert a new “General” STORAGE element named **StoWaitingRoom**.

**Step 2:** Entries (i.e., entities) are added to STORAGE elements via an *Insert* step and removed via the *Remove* step. Add two new processes named **WaitingRoomEntered** and **WaitingRoomExited**, as seen in Figure 8.34 and Figure 8.35, respectively. Specify the **StoWaitingRoom.Queue** as the *Queue State Name* property.



**Figure 8.34: Inserting into the WaitingRoom**



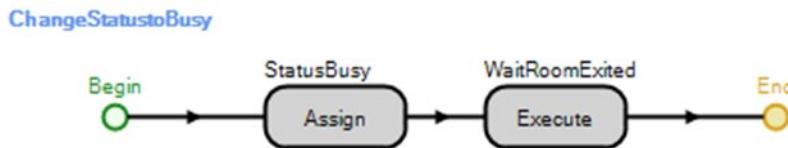
**Figure 8.35: Removing from the WaitingRoom**

**Step 3:** People enter the common waiting room when they leave the registration desk or when they are waiting at the driving test or the picture area.

- For the *After Processing* add-on process trigger of the **SrvRegistration**, specify the **WaitingRoomEntered** process.
- Select both the **SrvDrivingTest** and **SrvPicture** and then specify the **WaitingRoomEntered** process as the *Entered* add-on process trigger.

**Step 4:** People leave the waiting room when they enter service at the testers, driving test, or picture area.

- Select both the **SrvDrivingTest** and **SrvPicture**, then specify the **WaitingRoomExited** process as the *Before Processing* add-on process trigger.
- For the three testers (i.e., **SrvTests1**, **SrvTests2**, and **SrvTests3**), we will need to modify the **ChangeStatustoBusy** process since it is already specified as the *Before Processing* add-on process trigger. Insert an *Execute* step with the *Process Name* specified as **WaitingRoomExited**.

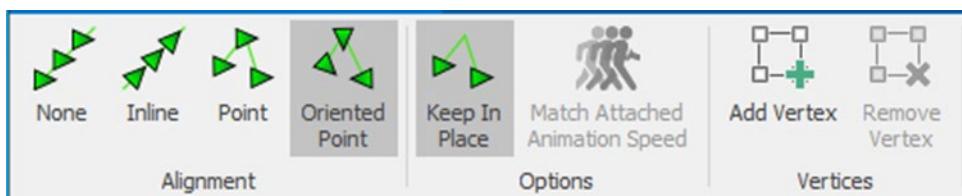


**Figure 8.36: Using Execute Step to Cause Applicants to Leave the Common Area**

**Step 5:** To “see” the waiting room, insert a **DetachedQueue** from the *Animation* tab. The *Queue State* property for the queue is the **StoWaitingRoom.Queue**. Draw a straight line for the queue.

**Step 6:** Most waiting rooms have people along the edge of a room rather than a straight line.<sup>110</sup>

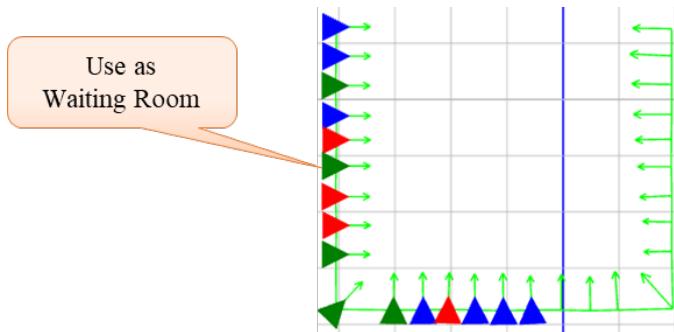
- Select the animated **WaitingRoom** queue, which is the green straight line by default.
- Change the *Alignment* to *Oriented Point* and then select the *Keep in Place* option since entities, once they sit, will not keep changing seats along the queue like a checkout line does, as seen in Figure 8.37.
- From the *Vertices* section, add two vertices by selecting the *Add Vertex* button and dragging the green vertex (●) onto the queue.



**Figure 8.37: Adding an Animated Queue**

<sup>110</sup> From an animation standpoint, we would delete the input buffer queues of the testing stations, driving test, and picture area.

- Form a U-shaped queue by moving the two new vertex points that were just added to the appropriate spots. Then continue adding vertexes and then moving the points inward to form the waiting room queue as seen in Figure 8.38.



**Figure 8.38: Adding an Animated Queue in a Different Shape**

**Step 7:** Save and Run the model.

**Question 37:** What is the average number and the maximum number in the waiting room (storage)?

---

**Question 38:** How much time, on average, is spent in that waiting area?

---

## Part 8.7: Using Appointment Schedules

In response to applicant waiting concerns, the office has decided to offer a few scheduled appointments to provide better service to the applicants. The office will still allow most people to “walk-in” but will prioritize those with an appointment. In particular, the office will schedule applicants for the following nine appointment times during the day: 8:30 am, 9:00 am, 9:30 am, 10:00 am, 2:00 pm, 2:30 pm, 3:00 pm, and 3:30 pm. The appointments are scheduled for applicants when there are lower arrival rates. Some applicants who have appointments will arrive early, while others may arrive late, which is called “early/late” arrivals. Also, it is expected that some may skip their appointment altogether (i.e., “no-shows”).

You need to realize that a “scheduled arrival process” is fundamentally different from the usual “random” arrival process in that the random arrival process has random interarrival times while the scheduled arrival process has known interarrival times. Randomness in the scheduled arrival process occurs in the early/late arrivals and the no-shows. In many “service industries,” scheduled arrival processes are more prevalent than random arrivals.

**Step 1:** The scheduled appointment times are specified via a data table. From the *Data* tab, add a new data table named **TableArrivals**. Insert a *Date Time* column named **ScheduledTime** from the *Standard Property* drop-down, as seen in Figure 8.39. Next, specify the scheduled appointments, which indicate the time of arrival.<sup>111</sup> At this point, these appointments are absolute arrival times (i.e., if the simulation was to start on 9/5/2015 instead of 9/6/2015, no arrivals would occur until the next day, or if the simulation start date was 9/7/2015, then no arrivals would occur). See the *Commentary* section at the end of the chapter on specifying a table with relative time offsets.

---

<sup>111</sup> Note, you can build the table in Excel as it can be easier and then copy the table into SIMIO.

	Scheduled Time
1	9/6/2015 8:30:00 AM
2	9/6/2015 9:00:00 AM
3	9/6/2015 9:30:00 AM
4	9/6/2015 10:00:00 AM
5	9/6/2015 2:00:00 PM
6	9/6/2015 2:30:00 PM
7	9/6/2015 3:00:00 PM
8	9/6/2015 3:30:00 PM

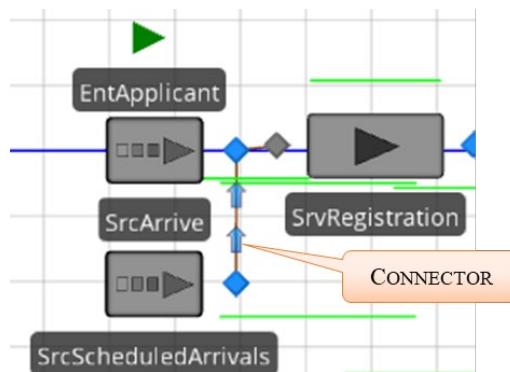
**Figure 8.39: Specifying the Appointments**

**Step 2:** Since we assume one person will arrive at the scheduled appointments, we will reduce the arrival rate table by one for the first and last four time periods so comparisons can be made, as shown in the partial table.

**Table 8.6: Hourly Arrival Rates for Applicants**

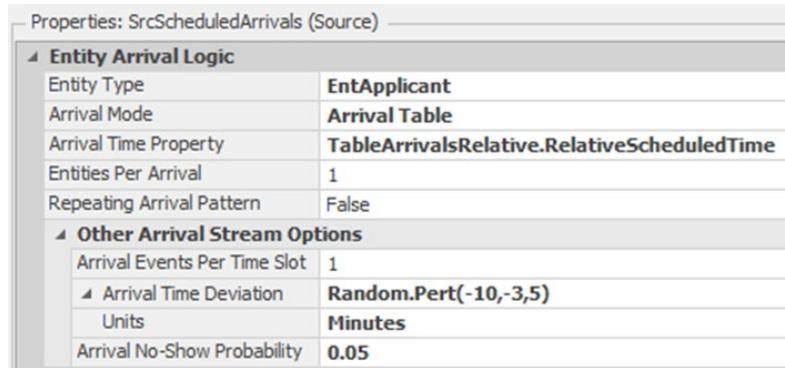
Time	Hourly Arrival Rate
8:30-9 am	14
9-9:30 am	9
9:30-10 am	7
10-10:30 am	11
<i>Middle Times Unaffected</i>	
2-2:30 pm	6
2:30-3 pm	4
3:00-3:30 pm	4
3:30-4 pm	4

**Step 3:** In the *Facility* tab, insert a new *SOURCE* object named **SrcScheduledArrivals**, as shown in Figure 8.40. Next, it is connected to the output node of the original *SOURCE* via a *CONNECTOR*, which allows arrivals to originate from one point and travel along the same five meter path to the reception area.



**Figure 8.40: Adding an Additional Source to Handle the Scheduled Appointments**

**Step 4:** To utilize appointment schedules, change the *Arrival Mode* to “**Arrival Table**” and specify the *Arrival Time Property* as the column **TableArrivals.ScheduledTime** as seen in Figure 8.41.



**Figure 8.41: Specifying the Appointment Scheduling in the SOURCE**

**Step 5:** The *Arrival Time Deviation* property is used to model the deviation from the appointed scheduled time specified by the **TableArrivals.ScheduledTime**. In this example, we are stating that applicants can arrive as early as ten minutes or as late as five minutes, but most will arrive three minutes early, using a Pert distribution<sup>112</sup>. See Table 8.7 for an explanation of all the appointment schedule properties.

**Table 8.7: Appointment Scheduled Properties**

Appointment Scheduled Property	Description
<i>Arrival Time Property</i>	The numeric table property specifies the list of scheduled arrival times, which can be a Date Time or any numeric or expression property.
<i>Arrival Events Per Time Slot</i>	This is the expected number of arrival events that will occur at each arrival time. You can think of this as how many batches will arrive, where the <i>Entities Per Arrival</i> will determine the number of Entities per batch that will arrive.
<i>Arrival Time Deviation</i>	It specifies the deviation from the scheduled time, which affects each batch differently. Therefore, two <i>Arrival Events Per Time Slot</i> and <i>1 Entities Per Arrival</i> will be different than one <i>Arrival Event Per Time Slot</i> and <i>2 Entities Per Arrival</i> . When a deviation or no-show probability is specified, as in the latter case, the two entities will always arrive at the same time.
<i>Arrival No-Show Probability</i>	Specify the probability of a no-show occurring, which is applied to a batch arrival.
<i>Entities Per Arrival</i>	The number of entities to create for each batch.
<i>Repeat Arrival Pattern</i>	When you reach the end of the table, determine whether or not to repeat the pattern. If you have specified actual times, these are converted to deviations or time offsets. For this example, the first arrival is at 12 pm, with the simulation starting at 11 am, which means there is a 1 hour time offset till the first arrival. So, at 4 pm, when the last scheduled arrival occurs, the simulation will start back up with arrivals at 5 pm to have the same offset if this property is set to "True."

**Step 6:** In this case, we assume 5% of the applicants are “no-shows” and skip their appointments. Therefore, set the *Arrival No-Show Probability* property to 0.05.

**Step 7:** In the **SrcScheduledArrivals** source reference, the process **SrcArrive\_CreatedEntity** is used as the *Created Entity* add-on process trigger. Also, in order to distinguish the scheduled arrivals, assign the

---

<sup>112</sup> Negative times mean the entity is early for their appointment, whereas positive number implies they are late.

`ModelEntity.Priority` a value “2” in the *Before Exiting* of the “State Assignments” section. Recall the default *Priority* state variable has a value of 1.<sup>113</sup> Refer to Figure 8.42.

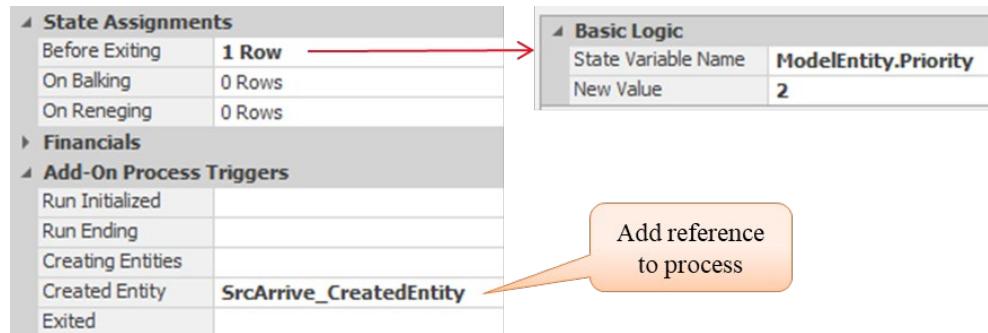


Figure 8.42: Specifying Properties of the SrcScheduledArrivals Source

**Step 8:** We would like to keep track of time in the system according to whether the applicant had a scheduled appointment or they were a walk-in. From the *Definitions→Elements* section, add two TALLY STATISTICS named **TallyStatSchedTimeInSystem** and **TallyStatWalkInTimeInSystem** with the *Unit Type* property set to “Time.”

**Step 9:** To obtain the time in system statistics for the two types of arrivals, each needs to be tallied. One way to obtain the tallies is to use the *Tally Statistics* section of the **Input@SnkLeave** specifying *On Exited*, as shown in Figure 8.43.

Basic Logic	
<b>Tally If</b>	<b>Custom Condition</b>
Condition	<b>ModelEntity.Priority == 1</b>
Tally Statistic Name	<b>TallyStatWalkInTimeInSystem</b>
Value Type	Expression
<b>Value Expression</b>	<b>ModelEntity.TimeInSystem</b>
Units	<b>Minutes</b>

Basic Logic	
<b>Tally If</b>	<b>Custom Condition</b>
Condition	<b>ModelEntity.Priority == 2</b>
Tally Statistic Name	<b>TallyStatSchedTimeInSystem</b>
Value Type	Expression
<b>Value Expression</b>	<b>ModelEntity.TimeInSystem</b>
Units	<b>Minutes</b>

Figure 8.43: Collecting Time in System by Arrival Type

**Step 10:** Select the experiment and add a new response for each of the TALLY STATISTICS named **WalkInTimeInSystem** with an *Expression* set to `TallyStatWalkInTimeInSystem.Average` and **ScheduledTimeInSystem** with `TallyStatSchedTimeInSystem.Average`. Both should have a *Unit Type* as **Time** with *Display Units* as **Minutes**.

**Step 11:** Save the current model. Reset the experiment and run for 100 replications.

*Question 39:* What is the average time in the system and half-width for each type?

---

*Question 40:* Is there any statistical difference in the times in the system for the applicant types?

---

**Step 12:** One of the potential issues is that scheduled patients are not processed differently in the model. Applicants who have scheduled appointments should have a higher priority than walk-in applicants.

- Therefore, select **SrvRegistration**, **SrvDrivingTest**, and **SrvPicture** SERVERS in the model and change their *Ranking Rule* property so that scheduled appointments have a larger priority, as seen in Figure 8.44.
- 

<sup>113</sup> The PRIORITY assignment could be done within the process **SrcArrive\_CreatedEntity**.

Process Logic	
Capacity Type	Fixed
Initial Capacity	
Ranking Rule	Largest Value First
Ranking Expression	Entity.Priority

Figure 8.44: Changing the Ranking Rule in the SERVERS

- For the testing SERVERS, the expression has to be a little more complicated because both entities and the MODEL seize server capacity. We need to give priority to the MODEL for the data entry process using the following expression seen in Figure 8.45. The Is.Entity function will return true if an applicant is requesting service and false otherwise.

Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	Largest Value First
Ranking Expression	Math.If(Is.Entity, Entity.Priority, 3)

Figure 8.45: Setting Priority Based on Entity and Model

**Step 13:** Save the model and rerun the experiment.

*Question 41:* What is the average time in the system and half-width for each arrival type?

---

*Question 42:* Is there an improvement in scheduled appointments?

---

**Step 14:** There does not seem to be any difference in time when we prioritized the applicants at each of the servers. Let's add a dynamic label to the entities to display the priority value and see if the model is working as predicted. Select the **EntApplicant** MODELENTITY and set the *Dynamic Label Text* property under the *Animation* section to display the priority as seen in Figure 8.46.

Animation	
Current Symbol Index	ModelEntity.Picture
Random Symbol	False
Current Animation Index	ModelEntity.Animation
Default Animation Action	MovingAndIdle
Link Segment Transition Type	Smooth
Draw Type	Single
Dynamic Label Text	ModelEntity.Priority

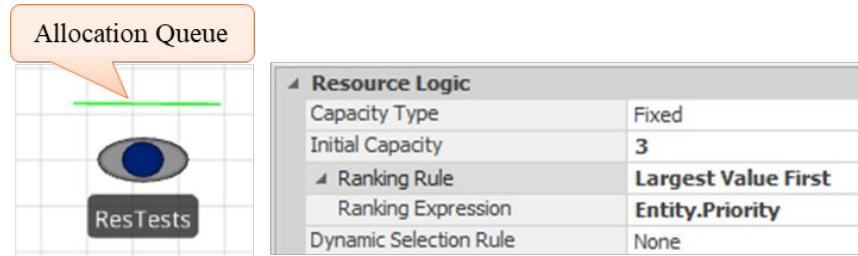
Figure 8.46: Adding a Dynamic Label that Moves with the Entity

**Step 15:** Run the model and observe the output buffer of the **SrvRegistration** around the 9:30 am time frame.

*Question 43:* Are all the scheduled applicants with priority two taken ahead of priority ones?

---

**Step 16:** The problem is there are three servers that have their own separate sorted queues when we really want one queue feeding those three servers to be sorted. The issue doesn't happen in the other areas, as the entities are all in the same queue and sorted properly. To fix the issue, insert a RESOURCE named **ResTests** with a capacity of three, and the *Ranking Rule* is set accordingly to act as this single queue. Also, select the **ResTests** and add the *ALLOCATION QUEUE* from the *Draw Queue* in the *Attached Animation* section.



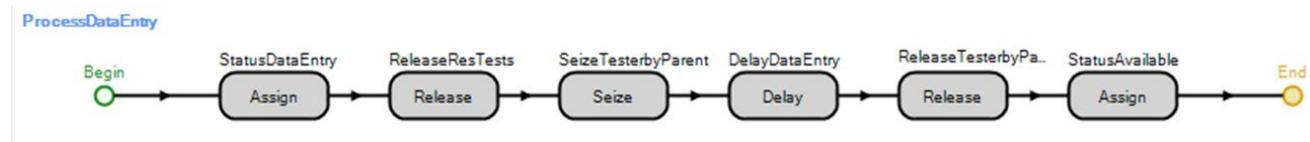
**Figure 8.47: Adding a Resource to Act as Single Queue Feeding Three Servers**

**Step 17:** Next, we need to seize the new **ResTests** once the applicants enter the output node of the **SrvRegistration** as they try to get service at testing. Select the output node of **SrvRegistration** (i.e., **Output@SrvRegistration**) and add an *Entered* Add-On Process Trigger, as seen in Figure 8.48. The **ResTests** will determine which applicant to serve next and allow the applicant to move on to the testers.



**Figure 8.48: Seizing the ResTests in the Output Node of the SrvRegistration**

**Step 18:** Once the entity finishes the testing, we need to release the **ResTests**. Insert a *Release* step in the **ProcessDataEntry** right after the status is updated, as seen in Figure 8.49.



**Figure 8.49: Releasing the ResTests After Finishing Testing**

**Step 19:** Save and run the model, observing whether or not the applicants are being ranked correctly now.

**Step 20:** Next, rerun the experiment.

**Question 44:** What is the average time in the system and half-width for each arrival type?

**Question 45:** Is there an improvement in scheduled appointments?

**Step 21:** The applicants are now being processed correctly. However, on further reflection, if a walk-in applicant has been in the system for more than 30 minutes, they should be given greater consideration/priority. Because the time in the system changes while the applicant is waiting in the input buffer, the static *Ranking Rule* property cannot be used as it orders the queue as the entities enter the queue and are not reordered later. On the other hand, the *Dynamic Selection Rule* is evaluated each time the SERVER capacity becomes available in order to choose the next applicant. Select the **SrvRegistration**, **SrvDrivingTest**, and **SrvPicture** SERVER objects and specify the “**Largest Value First**” as the rule as seen in Figure 8.50 with the *Value Expression* equal to

$(Candidate.Entity.Priority==2) \mid\mid (Candidate.Entity.TimeInSystem>MaxWaitTime)$ .

The expression will evaluate to either one (i.e., if the entity is a scheduled applicant or the entity has been in the system  $\frac{1}{2}$  hour) or zero otherwise.<sup>114</sup> Repeat for the **ResTests** RESOURCE object.

Ranking Rule	First In First Out
▲ Dynamic Selection Rule	<b>Largest Value First</b>
Value Expression	<b>(Candidate.Entity.Priority==2)    (Candidate.Entity.TimeInSystem&gt;0.5)</b>

**Figure 8.50: Specifying the Selection Rule to Prioritize Scheduled Applicants and Walk-ins**

**Step 22:** Save the model and rerun the experiment.

**Question 46:** What is the average time in the system and half-width for each arrival type?

---

## Part 8.8: Controlling the Simulation Replication Length

The number of applicants that seem to be in the system at the end of the 7.5 hours that the DMV is open is of concern. After more discussion with the DMV personnel, you realize that the office closes at 4:30 pm, but the people stay until the last applicant is served. It is easy to run the simulation for exactly 7.5 hours using the “Run Setup” section of the “Run” tab. However, we need the simulation to stop after serving the last person. This concern raises two modeling issues. First, how do we stop the arrivals after 4:30 pm? Second, how do we stop the simulation run after the last person has left the system? There are various ways to stop applicants’ arrival after 7.5 hours. For example, we could transfer them to a sink if the current time is after 4:30 pm (i.e., using selection weights on the paths). Another way would be to create a MONITOR that would cause a process to alter the interarrival time or the entities per arrival.

Note that the current **ArrivalRateTable** only has 15 time periods, representing 7.5 hours. If we extend the simulation run length beyond 7.5 hours, SIMIO will cycle back to the beginning of the table, repeating the values because they are relative offset values. The only way to force zeros is to extend the arrival table to contain 24 items (i.e., 12 hours) and set the remaining values to zero. A direct method to stop the arrivals is to use the “Stopping Conditions” within the **SOURCE**, which turns off arrivals based on either specifying a maximum number of arrivals<sup>115</sup>, some maximum time, or some stoppage event.

**Step 1:** Select the **SrcArrive** SOURCE and specify the *Maximum Time* property with a value of 7.5 under the *Stopping Conditions* section.



**Figure 8.51: Stopping Arrivals after 4:30**

**Step 2:** Set the run length to 12 hours and then run the model. You will notice that the applicants stop arriving at 4:30 pm, but the simulation continues displaying services (i.e., failures). A simulation run can be stopped using the *End Run* step. However, this step needs to be invoked under two conditions: (1) at 4:30 pm, in case no one is in the system, and (2) the more likely case when the last applicant is served after 4:30 pm.

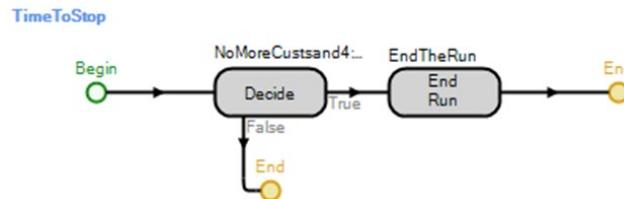
---

<sup>114</sup> If there is a tie (i.e., multiple entities have the same value), they are ordered based on First in First out rule.

<sup>115</sup> Note that this is the number of total “arrival events” and not the total number of entities arriving. One arrival event may yield more than one entity arriving.

**Step 3:** Let's first consider the case that the last applicant finishes after 4:30 pm when no more arrivals are permitted. Applicants exit the system through one of the three sink objects. So, at each of the sinks, we need to test whether this entity is the last applicant in the system and if it is after 7.5 hours. We will first create a general process.

- Go to the “Processes” tab and click on the “Create Process” button, naming the new process **TimeToStop**. Add a *Decide* and *End Run* step as in Figure 8.52.



**Figure 8.52: Check Last Applicant and End Run**

- The *Decide* step is “*ConditionBased*” on the following expression.

`(EntApplicant.Population.NumberInSystem <= 1) && (Run.TimeNow >= 7.5)`

This expression states that we will stop the replication if the current applicant being destroyed is the last applicant in the system and the simulation time is greater than or equal to 7.5 hours.

**Step 4:** Select the “*Destroying Entity*” add-in process trigger property for each of the three SINKS and specify the **TimeToStop** process for each.

**Step 5:** Within the *Run→Run Setup*, arbitrarily set the *Run Length* to 12 hours to be sure the simulation length does not end the run.

**Step 6:** Save and fast-forward the model to the end of the replication.

*Question 47:* At what time did the replication stop?

---

**Step 7:** If you make multiple replications, each replication will stop at a different time. Output statistics are appropriate for collecting statistics on the time the replications stop. Note that output statistics only collect one observation for each replication at the end of the run. Such statistics will allow us to determine how long the office stays open on average. From the *Definitions→Elements* section, insert a new OUTPUT STATISTIC named **OutStatOfficeCloses** with the expression `Run.TimeNow` and set the *UnitType* to “*Time*” and *Units* to “*Hours*.”

**Step 8:** Multiple replications of the model need to be executed to determine how long the office is open. From the “*Project Home*” tab, insert a new experiment named **ExpOfficeCloses**.

**Step 9:** Change the *Required* replications for the first scenario to 100.

**Step 10:** You need to add a response to evaluate each experiment scenario. Under the “*Experiment*” section, click the *Add Response* button to insert a response named **TimeAfterClosing**, which has an expression for the `OutStatOfficeCloses.Value - 7.5`. This will indicate when the simulation ended minus the 7.5 hours representing the amount of time over the normal day (see Figure 8.53).<sup>116</sup>

---

<sup>116</sup> Note we could have made the OUTPUT statistic expression be `Run.TimeNow - 7.5` instead.

General	
Name	TimeAfterClosing
Display Name	TimeAfterClosing
Description	
► Expression	OutStatOfficeCloses.Value - 7.5
▲ Unit Type	Time
Display Units	Minutes

**Figure 8.53: Specifying a Response for the Experiment**

**Step 11:** Save the project and execute the experiment, and from the Pivot grid, answer the following questions.

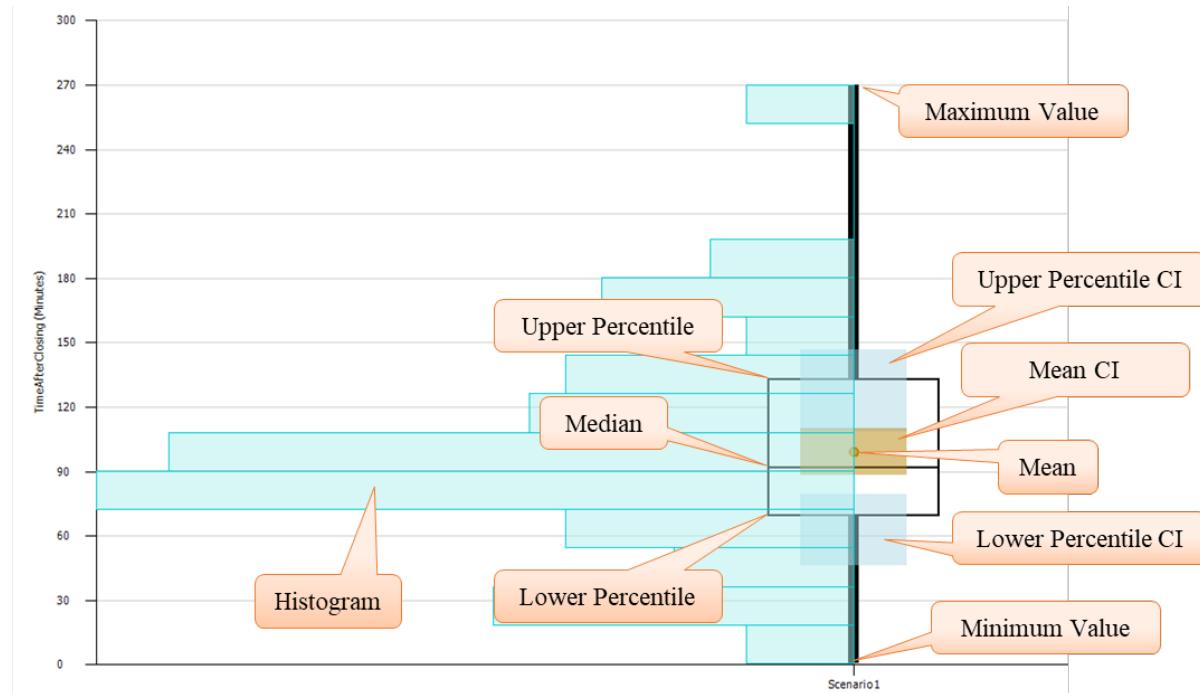
*Question 48:* On average, how long does the office stay open past the 7.5 hours?

---

*Question 49:* What is the half-width of the time when the office closes (in minutes)?

---

**Step 12:** Click on the “Response Results” tab to see the (SIMIO Measure of Risk & Error (SMORE) plot for the one response as seen in Figure 8.54. SMORE plots allow you to visualize the output response. It concisely displays the minimum, maximum, mean, median, lower, and upper percentiles,<sup>117</sup> as well as the confidence intervals of the mean and the lower and upper percentiles. The histogram of values is also displayed because the “Histogram” button is pushed.



**Figure 8.54: SMORE Plot with Histogram for the Additional Time the Office is Open**

*Question 50:* Based on the SMORE plot and its “Raw Data,” what are the mean and the 95% confidence intervals of the additional time office is open?

---

<sup>117</sup> By default it is the 25<sup>th</sup> and 75<sup>th</sup> percentiles which can be changed on the Experiment properties along with the default 95% confidence interval.

*Question 51:* What are the lower and upper percentiles and their values?

---

*Question 52:* What is the maximum additional time spent in the office, and does that make sense?

---

There are occasions when the maximum additional time spent is 7.5 hours (450 minutes) either via the SMORE plot or the results in the *Pivot Grid*. The simulation does not stop until the 15-hour run length is reached. Note an observation in a sink must occur before the stopping condition is checked.

*Question 53:* What happens if the last applicant leaves at 3:45 pm but no one arrives before 4:30 pm?

---

**Step 13:** In this situation, the last applicant leaves the system, but the current time is still less than 7.5 hours; therefore, the stopping condition is not invoked. The previous stopping method assumes that there would always be applicants in the system at 4:00 pm. The model needs to check to see if there are zero applicants at closing. Return to the Model, insert a new TIMER element named **TimerToStop** from the *Definitions→Elements* section, and set the *Time Interval* property to 7.5 hours<sup>118</sup>.

**Step 14:** In the *Processes* tab, insert a new process named **StoppingSimulation**, which utilizes the **TimertoStop** event as its *Triggering Event Name* property and the *Triggering Event Condition*, as seen in Figure 8.55. **The process will only run when the event occurs if the triggering event condition is also true.**

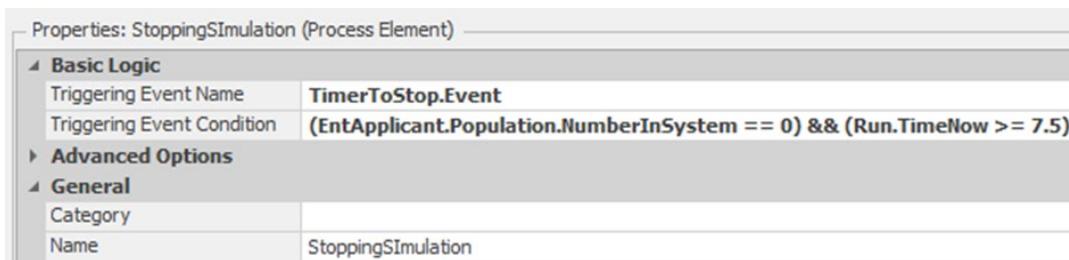


Figure 8.55: Specifying a New Process to be Executed when the Timer fires.

**Step 15:** Insert the *End Run* step, which will stop the simulation run as soon as the step is executed, as shown in Figure 8.56.

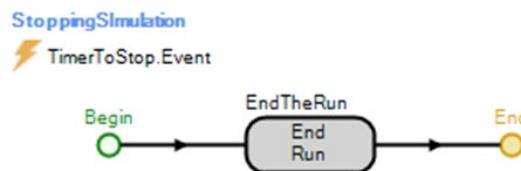


Figure 8.56: Stopping Simulation at Time 7.5 hours Process

**Step 16:** Save the model and rerun the experiment, looking at the “*Response Results*” again.

*Question 54:* On average, how long does the office stay open past the 7.5 hours?

---

*Question 55:* What is the maximum additional time spent in the office, and does that make sense?

---

<sup>118</sup> One timer event will occur at 0.0 since the *Time Offset* is 0.0, but it won't affect the simulation in this case.

*Question 56:* Is it realistic that the office should stay open only 90 minutes past its closing?

---

*Question 57:* What can be done to reduce the time the office is open after its closing time?

---

### Part 8.9: Commentary

- In Figure 8.39, the appointment schedule was designated using absolute dates and times. Internally, SIMIO converts the table into a time offset table based on the simulation starting date and time. One can also specify the appointment table directly as time offsets, as seen in Figure 8.57 Which represents the exact same table. The column type must be a numeric or an expression general property.

Table Arrivals Relative	
	Relative Scheduled Time
1	0
2	0.5
3	1.0
4	1.5
5	5.5
6	6.0
7	6.5
8	7.0

**Figure 8.57: Appointment table Using Time Offsets**

# Chapter 9

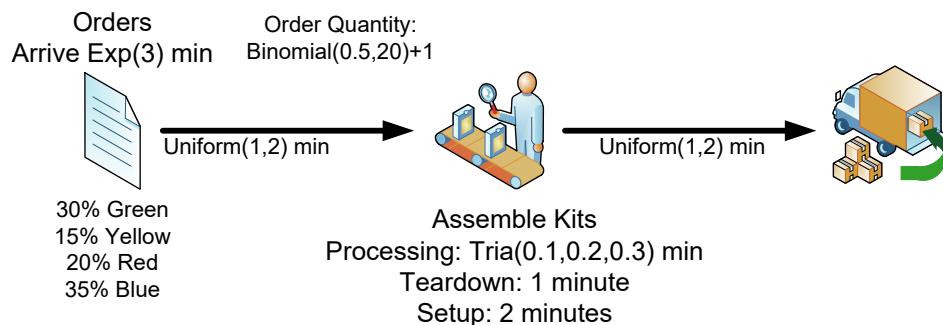
## The Workstation Concept: A Kitting Process

---

Often, a process, especially in a manufacturing environment, needs the ability to handle the situation of batching by processing an entity as though it were composed of several items or services one or more of the items at a time. Also, during the processing of the batches, we may need to consume or need materials to complete the processing. Materials are those items that are supplied to process and are directly required for the assembly/production of the finished item. A “bill of materials” typically determines what is needed for any operation. Materials are not objects but an inventory of items that can be consumed and replenished. While one can utilize entities and combiners to model this situation, it is not practical when you have millions of materials in the system. Also, processes often require a setup that may take time and resources or tear down/process finished time. In earlier versions of the book, the `WORKSTATION` object was used to model this scenario, but that object has now been deprecated and is no longer supported by SIMIO<sup>119</sup>.

### Part 9.1: The Kitting Process

A kitting process constructs a “kit” from available components. Orders arrive at a kitting workstation in a Poisson stream with an interarrival time of three minutes, as seen in Figure 9.1. An order is for a particular number of identical kits to be produced. The kits are assembled from three components: one SubA subassembly, one SubB subassembly, and two Connectors – this is also referred to as a Bill of Materials (BOM). The assembly of each kit has a processing time described by a triangular distribution with parameters of (0.1, 0.2, 0.3) minutes. Only one kit can be assembled at the kitting station at a given time. Information about the kits produced and components used is needed to analyze the operation, and information on the processing of the orders is also desired.



**Figure 9.1: The Kitting Operation**

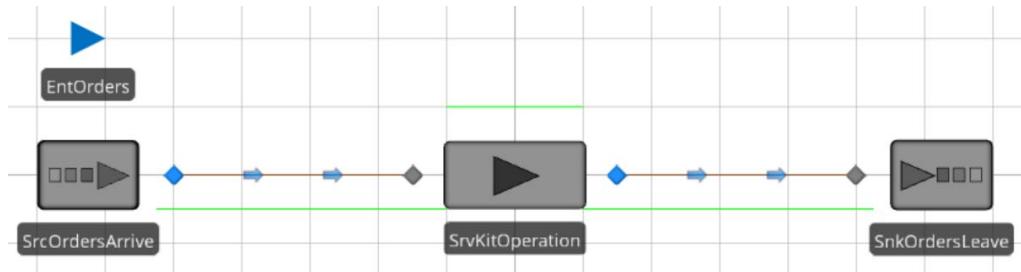
Four types of kits are produced from the same BOM, representing the following colors: 30% are green, 15% are yellow, 20% are red, and 35% are blue. Following each order that is processed, it takes one minute to tear down for the next kitting order and two minutes to setup for the following order. This assumption will be changed later. Finally, the orders will be assumed to take between one and two minutes to arrive at the kitting workstation as well as to travel to the exit from the kitting workstation.

---

<sup>119</sup> You can access deprecated objects by right-clicking in the Standard Library area.

**Step 1:** Create a new model that has one SOURCE (**SrcOrdersArrive**), one SINK (**SnkOrdersLeave**), and one WORKSTATION object (**WrkKitOperation**), as seen in Figure 9.2.

- Connect the objects using TIMEPATHS that take the appropriate Uniform (1, 2) minutes as the *Travel Time* property.
- The kits' orders arrive with an Exponential (3) minute interarrival time with random stream one.



**Figure 9.2: Kitting Operation Model**

**Step 2:** Insert a new MODELENTITY named **EntOrders** into the model.

**Step 3:** Add three additional symbols for the **EntOrders** entity by selecting the **EntOrders** instance and clicking the *Add Additional Symbol* button. Color the additional symbols “Yellow(1)”, “Red(2)”, and “Blue(3)”<sup>120</sup>. Note that symbols are numbered starting from zero.

**Step 4:** All of the SIMIO objects have specific built-in characteristics. Properties for an object (e.g., the initial number of in-system and maximum arrivals) are established at the time the object is created (usually at the beginning of the simulation) and cannot change during the simulation. State variables are characteristics that can change during the simulation. The SIMIO-defined *Priority* state variable<sup>121</sup> is an example of a characteristic often used for ranking. In SIMIO, you can add your own properties and discrete state variables to objects.

*Properties and States* are added through the “Definitions” tab of an object. All objects in SIMIO can have their own *Properties* and *States*. A state variable defined in the MODEL will only have one value and can be thought of as having a global scope. Defining a state variable on the MODELENTITY will allow each entity to have its own state variable value. Select the MODELENTITY in the [Navigation] panel and select the *Definitions→States* section. Each kit order will have a different quantity of kits requested, and a state variable is needed to track this information.

- Notice that SIMIO already defines two state variables. SIMIO defines **Picture** as a DISCRETE REAL STATE VARIABLE representing which picture should be displayed to represent the entity. SIMIO also declares the Animation state variable, a STRING STATE VARIABLE used to determine the type of moving animation the entity might do. The object also inherits other state variables from the parent object.

<sup>120</sup> Symbol numbers are in parentheses.

<sup>121</sup> There is an entity property called *Initial Priority*, which initializes the *Priority* state variable.

- You should add **EStaOrderQuantity** as a new Discrete Integer State Variable representing the number in the order, as seen in Figure 9.3.

Definitions Data		
Name	Object Type	Description
> State Variables (Inherited)		
State Variables		
Picture	Real State Variable	A utility sta
Animation	String State Variable	A utility sta
<b>EStaOrderQuantity</b>	Integer State Variable	

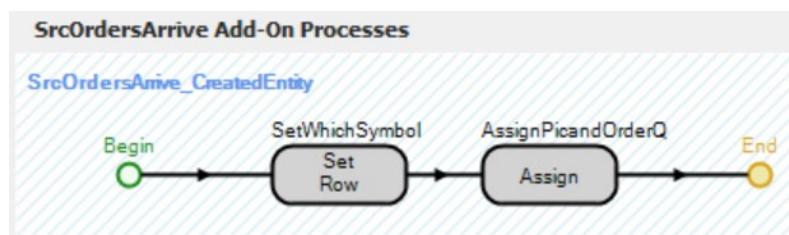
**Figure 9.3: Add a State Variable to the MODEL ENTITY Object**

**Step 5:** A table is needed to determine the percentage of time each type of order will occur and the symbol picture number. First, select the **MODEL** in the [Navigation] panel to return back to the simulation model. Let's create the **TableSymbol** for the entity pictures, as seen in Table 9.1 via the **Data** tab where the **Symbol** column should be an *Integer Standard Property*, the **Percent** could be an *Integer, Real, or Expression* property (i.e., numeric), and the **Color** is a *String Standard Property*.

**Table 9.1: Order Type Symbol and Percentage**

Table Symbol			
	Symbol	Percent	Color
1	0	30	Green
2	1	15	Yellow
3	2	20	Red
4	3	35	Blue

**Step 6:** Next, the order type (i.e., yellow, red, etc.), the order quantity for the arriving order, and the correct picture will be specified using an Add-On process trigger. Select the **SrcOrdersArrive** and double-click the *CreatedEntity* Add-On process trigger property, creating the process named **SrcOrdersArrive\_CreatedEntity**. After setting the table row, we can assign state variables to the incoming entities, as shown in Figure 9.4.<sup>122</sup>



**Figure 9.4: Setting the Row of the Data Table and Assigning the Picture.**

**Step 7:** Use the *Set Row* process step to randomly assign a row of the **TableSymbol** TABLE to the entity, which will be used in the *Assign* step to specify the entity's picture.

<sup>122</sup> Instead of using an add-on process trigger, the *On Created Entity Table Reference Assignment* could have been used to do the same thing along with the *Before Exiting State Assignments*.



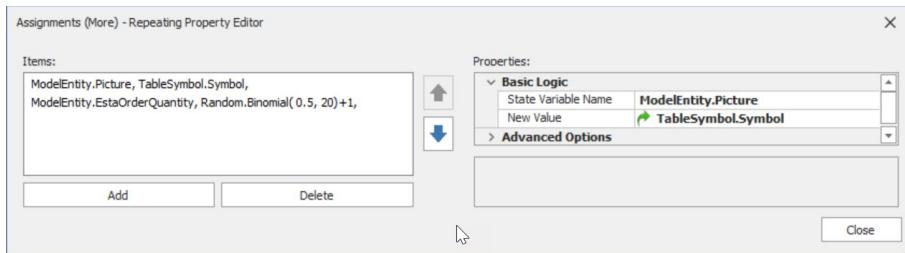
**Figure 9.5: Steps for the OrdersArrive\_CreatedEntity Process**

*Question 1:* Why was the table set in the “Created” Add-On Process Trigger versus “Creating,” as was done in Chapter 5?

---

**Step 8:** Next, assign the correct picture and the order quantity. You can do this in two different assignment steps or in one step using multiple rows by clicking the “0 Rows” button. You must add each assignment as seen below in Figures through Figure 9.8.

- Use the “Repeating Property Editor” to make multiple assignments, as shown in Figure 9.6.



**Figure 9.6: Making Multiple Assignments**

- Set the picture to represent the symbol associated with the specified order type, as in Figure 9.7.<sup>123</sup>



**Figure 9.7: Picture Assignment**

- Assign the order quantity from a Binomial Distribution as in Figure 9.8.



**Figure 9.8: Order Quantity Assignment**

*Question 2:* Note that a Binomial random variable is used to model the number of orders. Why are we adding one to the random variate?

---

**Step 9:** To assist in validating the model, select the entity and set the *Dynamic Label Text* to ModelEntity.EstaOrderQuantity which will display the order quantity as the model is running.

---

<sup>123</sup> The **Picture** state variable sets the symbol associated with the object in this case the ModelEntity.

**Step 10:** The time needed to process each kit at the workstation follows a Triangular (0.1, 0.2, 0.3) minute distribution. However, the workstation has a teardown time of one minute and a current setup time of a specific two minutes. We need to incur the setup time before processing the batch, followed by a teardown time after the batch is completed. In this current scenario, we need to add the setup and teardown times into the processing times.

Process Type	Specific Time
Processing Time	2+Random.Triangular(.1,.2,.3)+1
Units	Minutes

**Figure 9.9: Adding in the Setup and Teardown Time**

**Step 11:** Save your model with a new name and run the model for eight hours.

*Question 3:* What is the average flow time for the orders?

---

*Question 4:* How many orders were processed in the 8 hours?

---

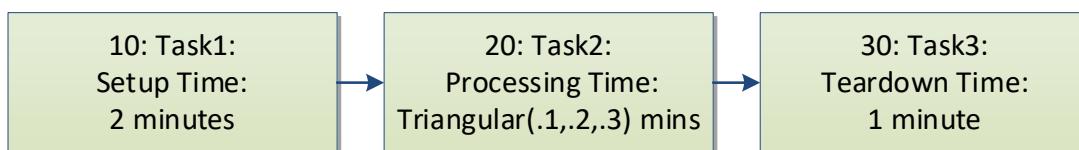
*Question 5:* What are we missing, or what assumptions are we making in the current model?

---

## Part 9.2: Modeling Batching, Setup, and Teardown Times

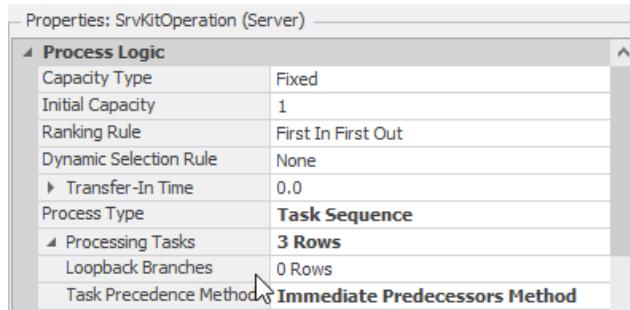
A service activity may be composed of several tasks rather than a simple processing time. For example, the kitting of an electronic product may employ a sequence of tasks to complete the kit (i.e., Setup, Processing, and Teardown). Likewise, an inspection may require several tests at the inspection station. Often, the details in these types of operations can be conveniently summarized by their processing time. There are, however, instances when the processing time needs a composition of a set of interrelated tasks. SIMIO, therefore, recognizes two types of processing stations within a SERVER, which are referred to as *Process Types*. Specifically, the alternative specifications are “*Specific Time*” or “*Task Sequence*”. We have only modeled using the Specific Time, the default value. A task sequence refers to a network of tasks ordered by precedence. For example, you need a nurse to take vitals and information, then you need to see the doctor, potentially followed by more nurse visits. This would be harder to handle using one SERVER. When more details about a process needed to be included in the model, the model was extended with more objects (i.e., additional servers were inserted). Using the *Task Sequence* process type, details about processing can be incorporated into a set of task sequences within a single SERVER rather than modeled with a series of SERVERS.

While the current scenario is a very simple process, separating out the setup and teardown steps into separate individual tasks, as seen in Figure 9.10, will allow more flexibility. For example, the setup and teardown may require an operator, sequence-dependent setups, set of tools, etc., to perform the task.



**Figure 9.10: Modeling Setup, Processing, and Teardown as Independent Processes**

**Step 1:** Change the *Process Type* of the **SrvKitOperation** to “Task Sequence.”



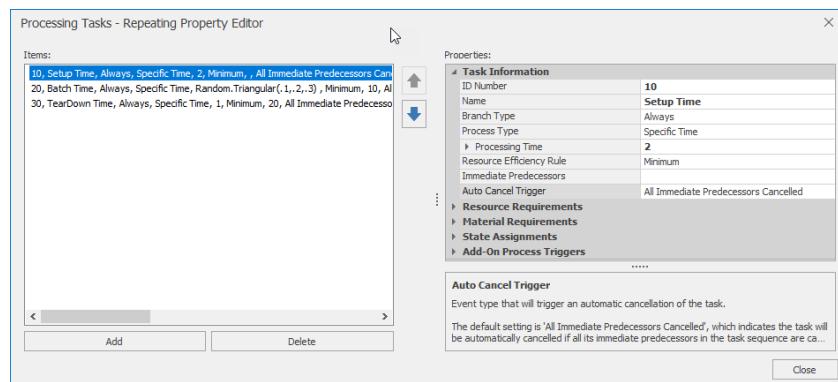
**Figure 9.11: Specifying the Task Sequence Process Type**

**Step 2:** We need the ability to specify the sequence order of a set of tasks (i.e., how do we indicate the precedence network). There are three different methods of specifying the precedence network, as outlined in Table 9.2. A more complicated task sequence diagram of the Bank Teller will be explored in detail in a later chapter. Change the *Task Precedence Method* to the **Immediate Predecessors Method**, which allows us to specify the predecessor tasks that must all be completed before the current task can start to be executed.

**Table 9.2: Task Precedence Methods**

Process Type	Description
Sequence Number Method	Expression represents the time needed for the task to complete.
Immediate Predecessors Method	Using the <i>Immediate Predecessors</i> property within each task, you specify all the tasks that need to precede this task before it is started (e.g., 10 or 10, 20).
Immediate Successors Method	Using the <i>Immediate Successors</i> property within each task, you specify all the tasks that need to start after this task is completed (e.g., 20 or 20, 30).

**Step 3:** Click on the *Processing Tasks* property additional items button [...] and add the three tasks using the specifics in Figure 9.12. The *ID Number* for each task must be a unique identifier that will be utilized to specify the *Immediate Predecessors*. Leave the *Immediate Predecessors* property blank if there are no predecessors, as is the case for the Setup Time task. For the batch time task, the preceding task is the setup time (i.e., identifier 10), while the batch time task (i.e., identifier 20) needs to be finished before starting the teardown task as specified.



**Figure 9.12: Specifying the Three Tasks**

Task Information	
ID Number	10
Name	<b>Setup Time</b>
Branch Type	Always
Process Type	Specific Time
Processing Time	2
Resource Efficiency Rule	Minimum
Immediate Predecessors	
Auto Cancel Trigger	All Immediate Pred

Task Information	
ID Number	20
Name	<b>Batch Time</b>
Branch Type	Always
Process Type	Specific Time
Processing Time	<b>Random.Triangular(.1,.2,.3)</b>
Resource Efficiency Rule	Minimum
Immediate Predecessors	10
Auto Cancel Trigger	All Immediate Predecessors Canc

Task Information	
ID Number	30
Name	<b>TearDown Time</b>
Branch Type	Always
Process Type	Specific Time
Processing Time	1
Resource Efficiency Rule	Minimum
Immediate Predecessors	20
Auto Cancel Trigger	All Immediate Pred

**Figure 9.13: Specifying the Setup, Batch, and Teardown Time Tasks**

**Step 4:** Save your model and run the model for eight hours.

*Question 6:* What is the average flow time for the orders?<sup>124</sup>

---

*Question 7:* How many orders were processed in the 8 hours?

---

**Step 5:** Up to this point, the processing time for each order was the same regardless of the order quantity. The processing time for a batch of parts to be processed individually is the sum of a series of independent processing times. One may think to model this situation is to sample from the random distribution once and then multiply by the order quantity (i.e., quantity \* triangular (.1,.2,.3)). Even though the mean will be equal, the variation will be much larger than if we summed up multiple random distributions. See Appendix A, section eight, on modeling the sum of  $N$  independent random variables. Change the processing time of the batch task using the `Math.SumOfSamples` function will sample from the distribution `EstaOrderQuantity` a number of times and sum the values.

Task Information	
ID Number	20
Name	<b>Batch Time</b>
Branch Type	Always
Process Type	Specific Time
Processing Time	<b>Math.SumOfSamples(Random.Triangular(.1,.2,.3),ModelEntity.EstaOrderQuantity )</b>
Units	Minutes
Resource Efficiency Rule	Minimum
Immediate Predecessors	10
Auto Cancel Trigger	All Immediate Predecessors Cancelled

**Figure 9.14: Handling the Batching of the Kits**

**Step 6:** Save your model and run the model for eight hours.

*Question 8:* What is the average flow time for the orders?

---

*Question 9:* How many orders were processed in the 8 hours?

---

<sup>124</sup> You may notice the answers to these two questions do not match the previous answers. This is due to the random number sampling order changing (i.e., different random numbers). If you run an experiment with multiple replications, the mean answers would be statistically the same. If one wanted to control random numbers (i.e., common random numbers), then we would need to set specific unique random streams for each distribution (e.g., `Random.Exponential(3, 1)`, which specifies the use of random stream number one for the distribution.

### Part 9.3: Sequence-Dependent Setup Times

Currently, the setup time is a constant of two minutes, regardless of the sequence of colors. Whenever the kitting workstation must change from one color to another, a set-up time depends on the color type that preceded the current order. The changeover time is shown in Figure 9.15. Let's add the sequence-dependent set-up times to our model.

	<i>From \ To →</i>	Green	Yellow	Red	Blue
Green		0	9	6	3
Yellow		3	0	6	3
▶ Red		3	9	0	3
Blue		9	9	3	0

Figure 9.15: Changeover Set-up Times (From/To) in Minutes

Under the **TASKSEQUENCE** properties, we specified the constant two-minute set-up time for the kitting operation using “Specific Time” as the *Process Type* with the *Specific* constant set-up (e.g., two minutes for the kitting operation), as seen in Table 9.3. But before we can use sequence-dependent set-up times, we need to know how the changeover times can be incorporated into the SIMIO model by using a changeover logic element and changeover matrix.

Table 9.3: Process Type Option Explanation

Process Type	Description
Specific Time	Expression represents the time needed for the task to complete.
Process Name	Specify a process that will be executed when the task starts. The task will finish once the process has finished executing.
SubModel	Allows you to create a different model within the current facility window. When the task starts, an entity of a specified type will be created and sent to the starting node of the submodel. You can save a reference to the original entity to pass information to the created entity. The task finishes once the created entity is destroyed by the submodel.
Sequence Dependent Set-up Time	Allows you to specify a sequence-dependent set-up logic.

**Step 1:** First, a **LIST** of identifiers is needed that can be used to create the changeover table (matrix), which, in this case, is a *List of Strings*. Under the “*Definitions*” tab, insert a **LIST** of Strings named **LstColors**. The contents of the **LstColors** list would correspond to our symbol colors: “Green,” “Yellow,” “Red,” and “Blue,” as seen in Figure 9.16. These must be put in this order to correspond to the symbols (0, 1, 2, and 3).

Name	
▲ Strings	
LstColors	
	String
0	Green
1	Yellow
2	Red
3	Blue

Figure 9.16: List of Colors

**Step 2:** Now, in the “Data” tab, add a CHANGEOVERS matrix (this is a “from-to” matrix) named **Changeover** and use the **LstColors** list for the associated list. Change the unit of time to *Minutes* and enter the values into the matrix from Figure 9.15.

**Step 3:** Next, from the “Definitions” tab, insert a CHANGE OVER LOGIC ELEMENT named **ChangeoverKitLogic**, as seen in Figure 9.17.

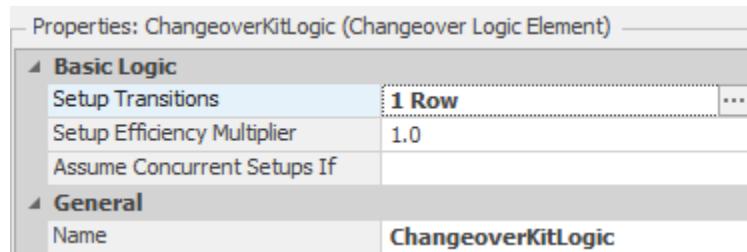


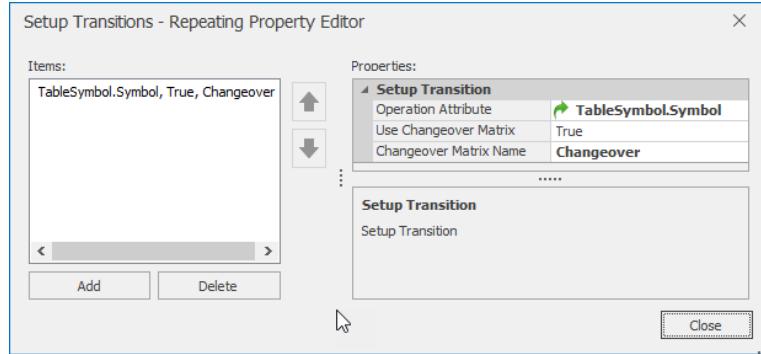
Figure 9.17: Inserting a Changeover Element

Table 9.4: Changeover Element Properties

Changeover Properties	Description
Set-up Transitions	Allows one to specify multiple set-up transitions either via change over the matrix or other criteria.
Set-up Efficiency Multiplier	It can be used to increase or decrease the total time taken for a set-up by either a constant or expression. It can be used to model variability in a constant set-up time.
Assume Concurrent Set-ups IF	An expression is evaluated before the change over time is calculated. If the condition is true, then the total time for the changeover will be the longest of several different set-up transitions. If it is blank or false, the total changeover time will be the sum of all the different set-up transitions.

**Step 4:** Click on the *Set-up Transitions ...* to invoke the repeating property editor to specify how to sequence-dependent transitions. Next, specify the setup time logic, as seen in Figure 9.18. The assigned symbol number will be used to determine the transition from the previous symbol, which is stored automatically by SIMIO.

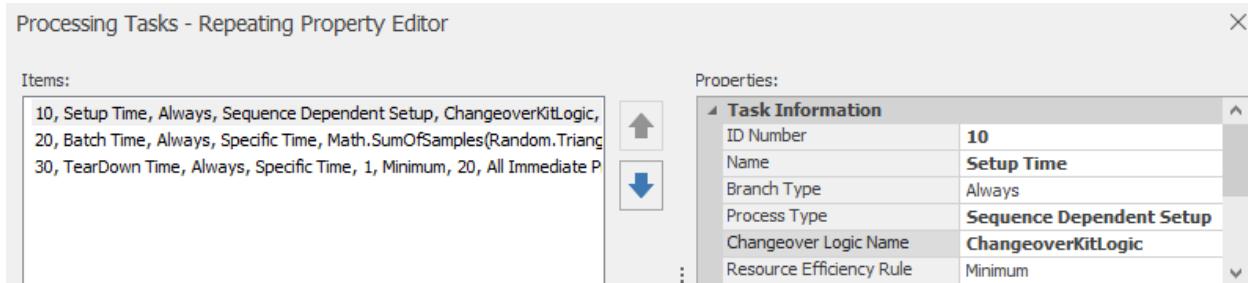
- *Operation Attribute:* TableSymbol.Symbol
- *Use Changeover Matrix property:* True
- *Changeover Matrix Name:* Changeover



**Figure 9.18: Specifying the Transitions as a Changeover Matrix**

**Step 5:** We can now finish by specifying the set-up time task sequence to use the **ChangeoverKitLogic** element. Select the **SrvKitOperation** and specify the other elements of the *Process Logic* for the **TASK SEQUENCE**, as seen in Figure 9.19.

- *Process Type* property: Sequence Dependent Set-up
- *Changeover Logic Name*: **ChangeoverKitLogic**

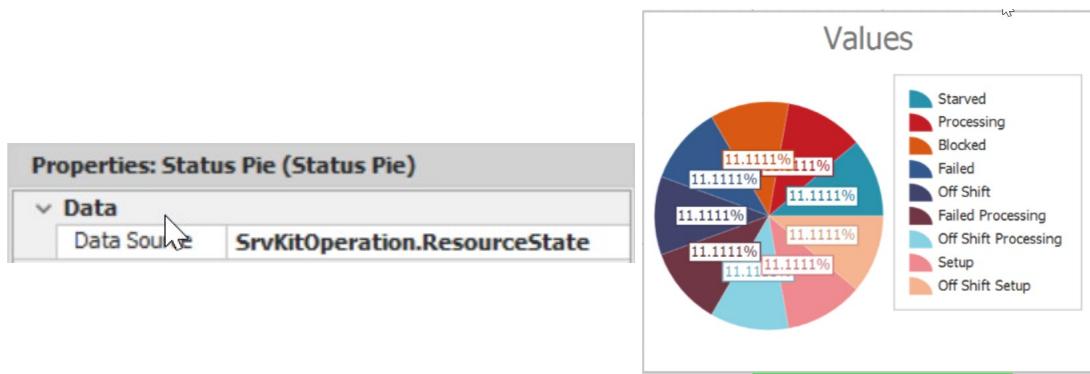


**Figure 9.19: Workstation Properties**

**Step 6:** Recall the SERVER can be in several different resource states, including *starved* (*i.e.*, *idle*), *processing*, performing a *set-up*, etc. To see all nine possible server states, click the **SrvKitOperation** and click the “Active Symbol” button. To see the percentage in these “resource” states evolve while the simulation is running, add a “*Status Pie*” from the *Animation* tab. Specify the *Data Source* property **SrvKitOperation.ResourceState**.<sup>125</sup>

---

<sup>125</sup> Note, if you select the **SrvKitOperation** object first and select the “*Status Pie*” from the “*Associated Animation*” tab, you will only need to specify **ResourceState**.



**Figure 9.20: Server Status Pie Chart**

**Step 7:** Save and run the simulation with animation for an hour. Then, run it in fast-forward mode for two, four, and eight hours. After eight hours, answer the following questions.

*Question 10:* What is the time in the system and the average set-up time of the server?

---

*Question 11:* How many orders were processed in the eight hours?

---

*Question 12:* What are we still missing in the current model?

---

#### Part 9.4: Sequence-dependent set-up Times that are Random

The previous section utilized the `CHANGEOVER` matrix to handle sequence-dependent set-up times. However, the matrix is limited to only constant/deterministic times and cannot handle the case when the set-up times have variability associated with them (i.e., random distributions). Since SIMIO's `CHANGEOVER` matrix cannot utilize distributions as the sequence-dependent set-up times, we will handle this manually using related data tables. Similar to previous examples, a child table will be used to define the set-up times for a particular part.

**Step 1:** Save the current model as a new name to be used again in a later section.

**Step 2:** First, we need to declare the relationship (i.e., primary key) for the parent table. From the `Data→Tables` section, select the column named **Color** in the **TableSymbol** table. Click the *Set Column as Key* button and fill in the field's values, as seen in Figure 9.21.<sup>126</sup>

---

<sup>126</sup> We could have used the **Symbol** field as the primary key, but this will make the setups easier to visualize in the related table.

Table Symbol   Table Change Overs			
	Symbol	Percent	Color
1	+	0	30
2	+	1	15
3	+	2	20
4	+	3	35

Figure 9.21: Adding the Color Field as the Primary Key

**Step 3:** Insert a new DATA TABLE named **TableChangeOvers**, as seen in Figure 9.22, which will be the child table.

Table Symbol   Table Change Overs			
	To Color	From Color	ChangeOver (Minutes)
1	Green	Green	0
2	Green	Yellow	3
3	Green	Red	3
4	Green	Blue	9
5	Yellow	Green	9
6	Yellow	Yellow	0
7	Yellow	Red	9
8	Yellow	Blue	9
9	Red	Green	6
10	Red	Yellow	6
11	Red	Red	0
12	Red	Blue	3
13	Blue	Green	3
14	Blue	Yellow	3
15	Blue	Red	3
16	Blue	Blue	0

Figure 9.22: Creating the Sequence Dependent Set-up Times Table

**Step 4:** From the *Add Column* section, select the *Foreign Key* to insert the **To Color** column. Set the *Table Key* property to **TableSymbol.Color** which will link the parent to the child changeover's table.

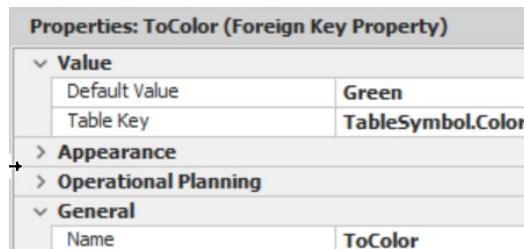
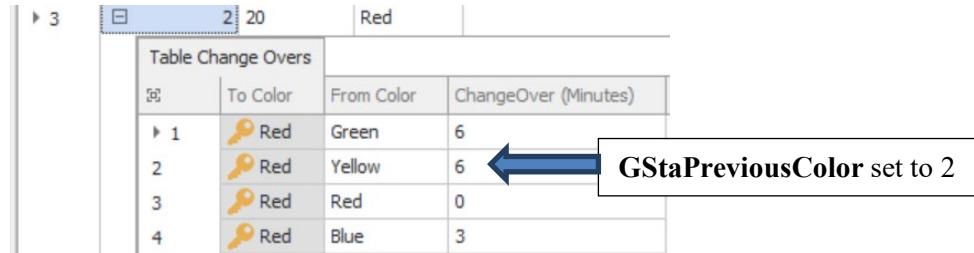


Figure 9.23: Adding the “To Color” Child Foreign Key

**Step 5:** Insert a *Standard String Property* named **From Color** and an *Expression Standard Property* named **ChangeOver**. Make sure to set the *Unit Type* property to “Time” and the *Default Units* to “Minutes.” Since this column is an *Expression* property column, we can specify any SIMIO expression, including sampling from a random distribution.

**Step 6:** Insert the values from Figure 9.22, which replicates the previous section's set-up times.

**Step 7:** When the **EntOrders** are created, they are assigned a row to the **TableSymbol** (see Figure 9.4), which was used to determine the picture. Now, they will also be assigned the related portion of the **TableChangeOvers**, as seen in Figure 9.24, if the new order was “Red.”



The screenshot shows a table titled "Table Change Overs" with four rows. The columns are labeled "Row", "To Color", "From Color", and "ChangeOver (Minutes)". The data is as follows:

Row	To Color	From Color	ChangeOver (Minutes)
1	Red	Green	6
2	Red	Yellow	6
3	Red	Red	0
4	Red	Blue	3

A blue arrow points from the text "GStaPreviousColor set to 2" to the "From Color" column of the second row (row 2).

Figure 9.24: Related Records in the ChangeOver Table Associated with Red Orders

**Step 8:** If the new order “Red” is next to be kitted and the previous order was “Yellow,” then the set-up time should be six, which is associated with the second row of the related table, as seen in Figure 9.24. However, we need to know which color was processed last. From the “Definitions” tab, insert a new MODEL *Discrete Integer State variable* named **GStaPreviousColor** with a default value of “1,” indicating the last color processed.

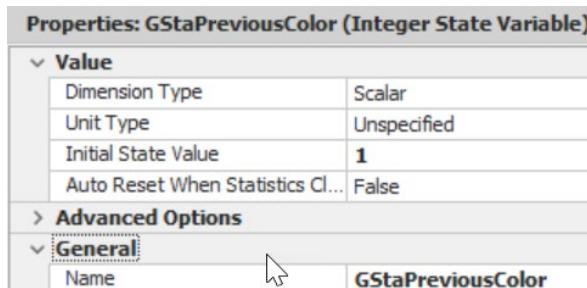
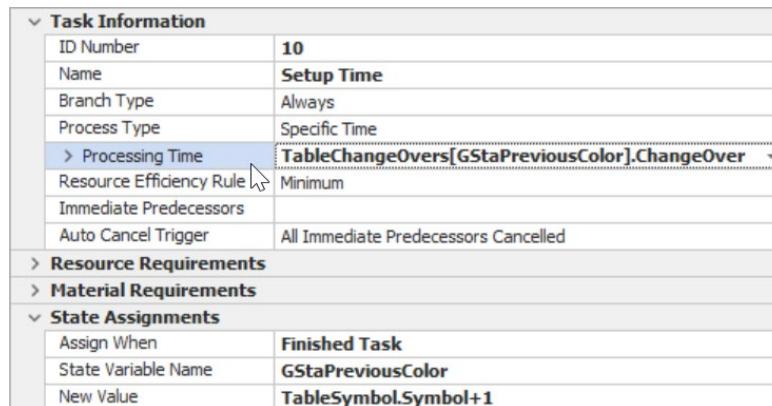


Figure 9.25: Insert a State Variable to Keep Track of the Last Order Processed

**Step 9:** Select the “Set-up Time Task” from the TASK SEQUENCES of the **SrvKitOperation**. Under the *State Assignments* property, set the **GStaPreviousColor** to the current symbol plus one, which will be used for the next order. The symbols range from zero to three, but table rows start at one. Change the *Assign When* properties to “Finished Task” will make the assignment after the set-up task finishes.



The screenshot shows the "Task Information" dialog for a task with ID 10, named "Setup Time". Under "Processing Time", the formula is set to "TableChangeOvers[GStaPreviousColor].ChangeOver". In the "State Assignments" section, the "Assign When" is "Finished Task", "State Variable Name" is "GStaPreviousColor", and "New Value" is "TableSymbol.Symbol+1".

Figure 9.26: After Set-up is Completed, Set the Previous Color

**Step 10:** Next, set the *Process Type* property back to “**Specific Time**” and specify the *Processing Time* to be `TableChangeOvers[GStaPreviousColor].ChangeOver`. The **GStaPreviousColor** indexes into the related table to specify the set-up time.

**Step 11:** Save and run the model comparing the results from the previous section.

*Question 13:* What is the time in the system and the average set-up time of the workstation?

---

**Step 12:** To demonstrate the ability to have variable set-up times, change all the set-up times from nine minutes to `Random.Pert(8, 9, 10)` minutes. Save and run the model comparing the time in the system and the average set-up time to the previous results.

*Question 14:* What is the time in the system and the average set-up time of the workstation now?

---

## Part 9.5: Using Materials in the Kitting Operation

SIMIO can keep track of raw materials that must be consumed, as specified in the “bill of materials” needed to create a part or complete an assembly. If the raw material is unavailable, parts (i.e., entities) cannot be processed and will wait until the required materials become available. These can be used to track inventories much easier than if you were just using state variables. In this example, each kit needs three materials in order to be produced.

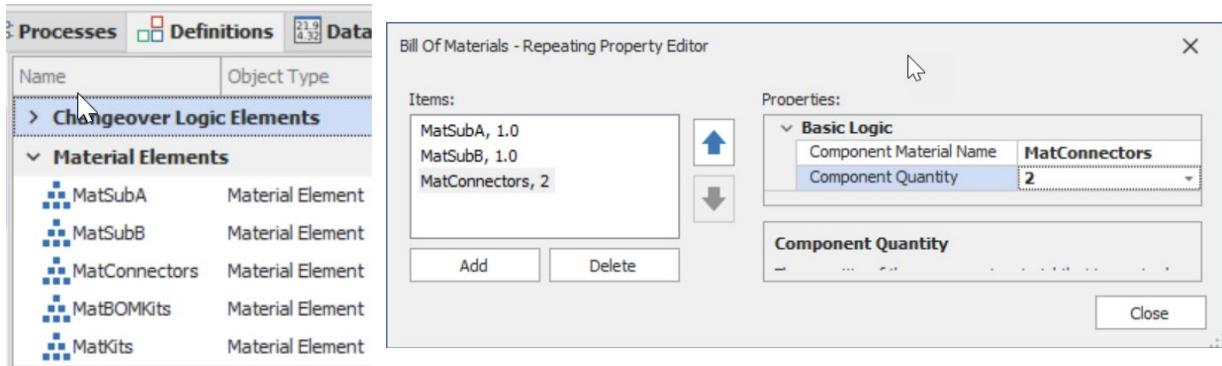
**Step 1:** Return to the model from Part 9.3: of this chapter and save it as a new model.

**Step 2:** First, we need to declare the existence of raw materials that make up a kit and the Bill of Materials that specify the number of each raw material type needed to make a kit. Five new MATERIAL “Elements” need to be defined via the *Definitions→Elements→Workflow* section, as seen in Figure 9.27.<sup>127</sup>

- Create **MatSubA**, **MatSubB**, and **MatConnectors** as materials types, all with *Initial Quantities* of zero, which will be changed shortly.
- Next, create a **MatBOMKits** as a MATERIAL element, but using the *Bill of Material* Repeating Property Editor as shown in Figure 9.27 to define what materials make up this product (i.e., one **MatSubA** and **MatSubB** and two **MatConnectors**).

---

<sup>127</sup> Elements are objects that represent things in a process that change state over time (e.g., TALLYSTATISTIC, STATESTATISTIC, TIMER, etc.)



**Figure 9.27: Five Materials and the Bill of Materials for the MatBOMKits**

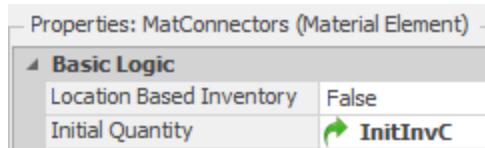
- Finally, create **MatKits** as the last “Material” element (with an *Initial Quantity* of 0) which will represent the finished product produced from the raw materials in the bill of materials.

**Step 3:** Since the initial inventory is a fixed characteristic of our model and only needs to be initialized at the start of the simulation, here is an ideal place to use a SIMIO Model “property.” From the *Definitions→Properties* section, add new properties to objects. Create three new “Standard Real or Expression Properties” named **InitInvA**, **InitInvB**, and **InitInvC**. Place all of these properties into a new **InvModel Category**. The first time, the new category name (**InvModel**) needs to be typed into the window. You can select the category from the drop-down menu for the next two properties.



**Figure 9.28: Creating Initial InventoryProperties**

**Step 4:** Return to the material “Elements” and set the *Initial Quantity* property for each of the three materials (i.e., **MatSubA**, **MatSubB**, and **MatConnectors**) as a reference property to the previously defined properties. Remember, this is done by right-clicking on the drop-down arrow and selecting the referenced property, as seen in the figure below.



**Figure 9.29: Specifying the Initial Inventory of Materials using the Inventory Properties**

**Step 5:** We can use the materials as part of the definition of the task sequences or use a process to consume and produce materials. Now return to the model under the *Facility* tab, and for the SERVER **SrvKitOperation** TASK SEQUENCES, fill in the *Materials Requirements* properties (i.e., *Action Type*, *Material Name*, and *Quantity property under the Required Quantity & Constraints*) as seen in Figure 9.30. The *Quantity property* specifies how many materials to consume or produce. We will use the “Batch Time” task to consume materials based on the bill of materials and the “TearDown Time” task to produce the kits.

Material Requirements	
Action Type	Consume
Consumption Type	<b>Bill Of Materials</b>
Material Name	<b>MatBOMKits</b>
Inventory Site Type	None
Required Quantity & Constraints	
Quantity	<b>ModelEntity.EstaOrderQuantity</b>
Lot ID	

Material Requirements	
Action Type	<b>Produce</b>
Production Type	Material
Material Name	<b>MatKits</b>
Inventory Site Type	None
Required Quantity & Constraints	
Quantity	<b>ModelEntity.EstaOrderQuantity</b>
Lot ID	

Batch Time Task

TearDown Time Task

Figure 9.30: Other Material Requirements for a WorkStation

**Step 6:** To access the new MODEL properties we created, click on the Model “Properties” (by right-clicking the “Model” object in the [Navigation Panel]).<sup>128</sup> In the **Controls → InvModel** section set the values of the **InitInvA** to 100, **InitInvB** to 100, and **InitInvC** to 200.

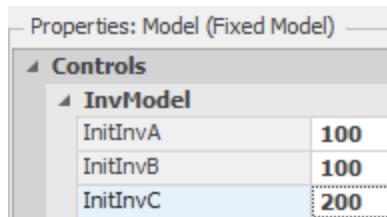


Figure 9.31: Setting Model Properties

**Step 7:** Save and run the new model for eight hours, observing the statistics. While you may want to run it for a while looking at the animation, you will need to fast-forward to the end to get the results.

*Question 15:* After looking at the Kits produced, do you have concerns about the model specifications?

---

*Question 16:* How many kits were produced in the eight hours?

---

*Question 17:* How many orders were satisfied over the eight hours?

---

*Question 18:* What was the cycle time for the orders?

---

*Question 19:* What was the average **MatSubA**, **MatSubB**, and **MatConnectors** *Quantity In Stock*?

---

## Part 9.6: Raw Material Arrivals during the Simulation

Instead of material being available for the entire eight-hour day at the beginning of the simulation, the raw materials will be scheduled to arrive every two hours, beginning at time zero. When the stock arrives, the

---

<sup>128</sup> Note you can change the name of the fixed model as well as set other model properties.

inventory is returned to a target stock level – implementing an “order-up-to-inventory policy.” How would this be modeled in SIMIO?

**Step 1:** Save and run the model for eight hours using initial inventory quantities of 1000, 1000, and 2000 for **SubA**, **SubB**, and **Connectors** (specified in the “Model Properties”).

*Question 20:* What was the average **MatSubA**, **MatSubB**, and **MatConnectors** *Quantity In Stock*?

---

*Question 21:* How many kits were produced?

---

**Step 2:** Now, let’s implement a replenishment system in which a stocker comes by the kitting station every two hours, beginning at time zero, and fills up the stock for **MatSubA**, **MatSubB**, and **MatConnectors** to particular quantities.

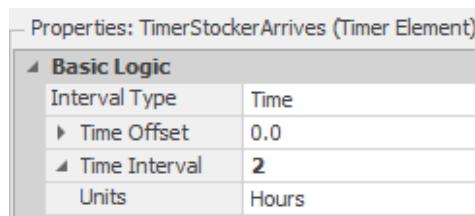
**Step 3:** Add three order quantity properties for the model under the *Definitions* tab: **OrderQtySubA**, **OrderQtySubB**, and **OrderQtyConn**. Each of these properties should be an *Integer Data Format* added to the **InvModel** category, as shown in Figure 9.32. These properties will be used as the order-up-to quantities that the stocker uses to replenish the raw material inventory.<sup>129</sup>



**Figure 9.32: Order Quantity Properties**

**Step 4:** Under the *Definitions→Elements* section, add a **TIMER** element that will fire an event based on a time interval (a “timer event” will be an interruption in the simulation that allows a process to execute). See Figure 9.33.

- Start the timer at time 0.0 (i.e., *Time Offset* property should be 0), meaning the first event fires at time zero, and it then should “fire” an event every two hours (i.e., *Time Interval* should be “2”).
- Name the timer **TimerStockerArrives**.



**Figure 9.33: Timer for Stocker Arrivals**

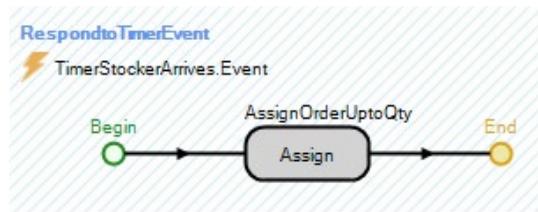
**Step 5:** Under the “*Processes*” tab, create a process named **RespondtoTimerEvent** that will respond to the timer event when it expires, as seen in Figure 9.34.

---

<sup>129</sup> Create one of the properties with the correct category and name. Then copy, paste it and change the name appropriately.

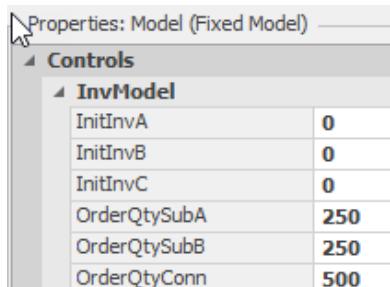
- *Triggering Event* property: **TimerStockerArrives.Event**
- Add an *Assign* step that makes the following assignments to the process, which always makes the quantity available equal to the order quantity (i.e., order-up-to policy).<sup>130</sup>

```
MatSubA.QuantityInStock = OrderQtySubA
MatSubB.QuantityInStock = OrderQtySubB
MatConnectors.QuantityInStock = OrderQtyConn
```



**Figure 9.34: Process Used to Respond to the Timer Event**

**Step 6:** Now, in the “Model Properties,” set the initial values of the raw materials **InitInvA**, **InitInvB**, and **InitInvC** elements to zero since the first **TimerStockerArrives** event occurs at time 0.0 and will replenish the stock at the beginning of the simulation. Set the order quantities for **OrderQtySubA**, **OrderQtySubB**, and **OrderQtyConnectors** to 250, 250, and 500, respectively.



**Figure 9.35: Setting the Model Properties for Order Quantities and Initial Inventory**

**Step 7:** Save and run the model for eight hours and answer the following questions.

**Question 22:** What are the average **MatSubA**, **MatSubB**, and **MatConnectors** inventory levels?

---

**Question 23:** How many kits were produced during the eight hours?

---

## Part 9.7: Implementing a Just-In-Time Approach

The previous example assumed a stocker would come by (exactly) every two hours and replenish the inventory up to a certain level (i.e., an order-up-to policy). However, the company is moving toward a

---

<sup>130</sup> SIMIO provides a *Produce* step that essentially does the same thing and a *Consume* step which would perform a subtraction, but the logic using an *Assign* step seems just straight forward. In this example, you would add three *Produce* steps that would produce for example  $\text{OrderQtySubA} - \text{MatSubA.QuantityAvailable}$ .

more Just-In-Time (JIT) type of operation. They have a supplier that is located a distribution warehouse very near the plant and will respond to orders very quickly. The company has implemented a monitoring system such that when the inventory drops below a certain threshold (i.e., reorder point), an order is placed to the supplier, who can supply the products within an hour.

**Step 1:** Save the current model to a new name. You need to delete the **TimerStockerArrives** **TIMER** Element and the **RespondToTimerEvent** process or disable the **TIMER** by setting the *Initially Enabled* property to **False** under the *General* properties of the **TIMER**.

**Step 2:** Add a new **SINK** named **SnkInventory**. Position it underneath the Kit Operation, which will be used to model the arrival of new raw materials (**MatSubA**, **MatSubB**, and **MatConnectors**).

**Step 3:** Add three reorder point properties for the model under the “*Definitions*” tab named **ReorderPTSubA**, **ReorderPTSubB**, and **ReorderPTConn**. Each property should be an “*Integer*” *Data Format* added to the **InvModel** category. These properties will be used as the reorder points for the JIT system to replenish the raw material inventory.

**Step 4:** Go to the “*Definitions*” tab and add three **MONITOR** elements underneath the “*Element*” section named **MonitorA**, **MonitorB**, and **MonitorC**. A monitor element can be used to monitor the value of a state variable when it changes value or when the state variable crosses some threshold (either from above or below). Refer to Figure 9.36 for the **MONITOR** properties for **MonitorA**.

Properties: MonitorA (Monitor Element)	
Basic Logic	
State Variable Name	<b>MatSubA.QuantityInStock</b>
Monitor Type	<b>CrossingStateChange</b>
Crossing Direction	<b>Negative</b>
Initial Threshold Value	 <b>ReorderPTSubA</b>

**Figure 9.36: Monitor Properties**

The following settings for **MonitorA** will monitor the **SubA** inventory level.

- *Monitor Type* should be *CrossingStateChange*.
- *Crossing Direction* should be *Negative* (i.e., 5 to 4 will fire an event if 4 was the threshold).
- The *Threshold Value* should be set to the new reference property named **ReorderPTSubA**.
- The **MONITOR** can cause a process to fire just as the **TIMER** element did, but we will not use this feature this time.

**Step 5:** Repeat the same process for raw material **MatSubB** and **MatConnectors** material using **ReorderPtSubB** and **ReorderPtConn** reference properties, respectively.

**Step 6:** Add three **SOURCES** (i.e., suppliers) named **SrcSupplierA**, **SrcSupplierB**, and **SrcSupplierC**, which will send the various raw materials to the kitting operation, as seen in Figure 9.37.

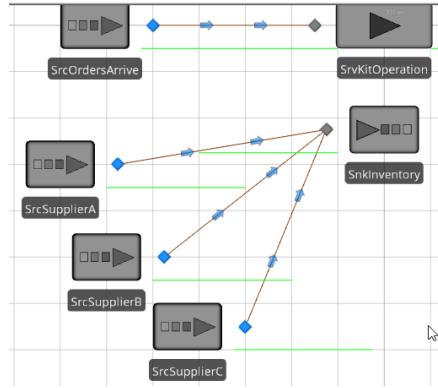


Figure 9.37: Supplier Sources

- Assume that it uniformly takes between 50 and 65 minutes for the raw materials to be delivered between the SOURCES and the **SnkInventory** SINK (this is modeled as the *Travel Time property of a TIMEPATH*).
- Name each TIMEPATH as **TPA**, **TPB**, and **TPConn** for the three TIMEPATHS, respectively.
- Set the *Arrival Mode* property of the suppliers to *On Event* and the *Event Name* to `MonitorA.Event`, `MonitorB.Event` and `MonitorC.Event` respectively for each of the three sources as seen for **SrcSupplierA** in Figure 9.38. Each source will create an entity when its monitor event fires and sends the replenishment entity to the kitting operation.

Properties: SrcSupplierA (Source)	
Entity Arrival Logic	
Entity Type	<b>EntOrders</b>
Arrival Mode	<b>On Event</b>
Initial Number Entities	0
Triggering Event Name	<b>MonitorA.Event</b>
Triggering Event Count	1
Entities Per Arrival	1

Figure 9.38: Utilizing an On Event Arrival Process

**Step 7:** When the raw material reaches the end of the path, you need to increment the current quantity by the appropriate order quantity using the *ReachedEnd* add-on process trigger for each of the three TIMEPATHS. A *Tokenized Process* will be used instead of three separate processes. From the *Definitions→Token* section, add a custom TOKEN named **TknReplenish**. For this TOKEN, add a “*Material Element Reference*” state variable named **TStaWhichMaterial** to reference the correct material and an “*Integer*” state variable **TStaQuantity**, as seen in Figure 9.39

Name	
<b>Tokens</b>	
TknReplenish	
Name Object Type	
<b>State Variables</b>	
ReturnValue	Real State Variable
TStaWhichMaterial	Material Reference State Variable
TStaQuantity	Integer State Variable

Figure 9.39: Adding the Custom TOKEN to Pass in the Material and Quantity

**Step 8:** From the *Processes* tab, insert a new process named **Replenish**, which uses the **TknReplenish** and specifies two *Input Arguments* properties named **MaterialToProduce** and **QuantityToProduce**, as seen in Figure 9.40.

Properties: Replenish (Process Element)	
<b>Basic Logic</b> <b>Advanced Options</b>	
Token Class Name	TknReplenish
Input Arguments	2 Rows
Return Values	0 Rows
Token Action On Associated...	ContinueProcess
Token Action On Associated...	ContinueProcess

Input Arguments - Repeating Property Editor	
Items:	MaterialToProduce , , TknReplenish.TStaWhichMaterial QuantityToProduce , , TknReplenish.TStaQuantity
	Properties: <b>Basic Logic</b> Name: MaterialToProduce Description: State Variable Name: TknReplenish.TStaWhichMaterial

Figure 9.40: Specifying the Custom Token and Input Arguments

**Step 9:** Insert a *Produce* step (see Figure 9.41), which will be used to add the order quantity to the particular material passed to the process. You will need to type the material name property value **TknReplenish** physically.**TStaWhichMaterial** directly rather than selecting it from the drop-down list.

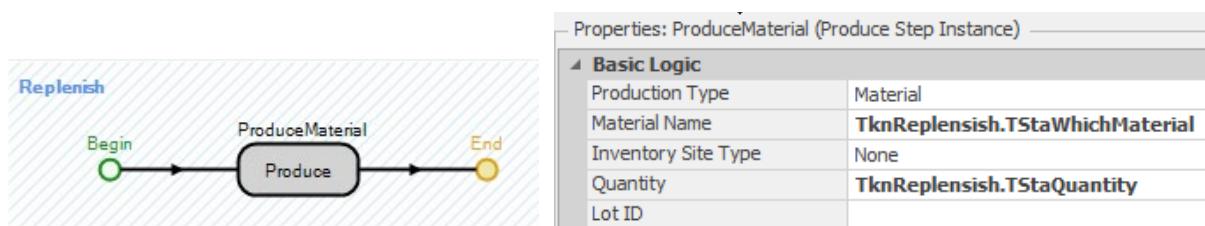
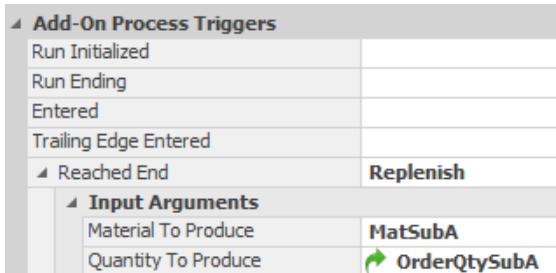


Figure 9.41: Incrementing the Quantity Available

**Step 10:** For each of the **TIMEPATHS** (i.e., **TPA**, **TPB**, and **TPC**), specify the **Replenish** process as the *ReachedEnd* Add-On Process Trigger property using the appropriate material and order quantity. Figure 9.42 shows the example for **TPA** using material **MatSubA** and quantity **OrderQtySubA**.



**Figure 9.42: Using the Tokenized Process for Material A Production TPA Path**

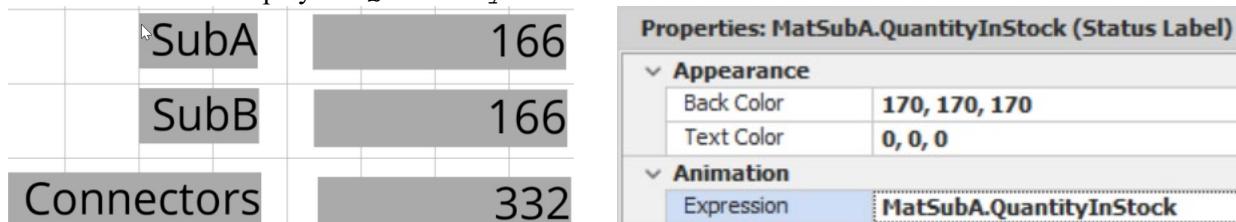
**Step 11:** In the Model properties, set the initial values of the raw materials **MatSubA**, **MatSubB**, and **MatConnectors** elements back to 250, 250, and 500 since the replenishment is being triggered. Set the reorder points and order quantities to the values provided in Table 9.5.

**Table 9.5: Initial Inventory, Reorder Point, and Order Quantities**

Raw Material	Reorder Point	Order Quantity	Initial Inventory
SubA	125	200	250
SubB	125	200	250
Connector	250	400	500

**Step 12:** While the simulation is running, we would like to display the available quantities of each of the three raw materials: **MatSubA**, **MatSubB**, and **MatConnectors** with either status labels or table.

- Add six “Status Labels” from the *Animation* tab that will display the available quantities of each of the three raw materials: **MatSubA**, **MatSubB**, and **MatConnectors**, perhaps as seen in Figure 9.43. The first three status labels are just text, with the other three labels having an expression, which will display the *QuantityInStock* for each material.



**Figure 9.43: Status Labels and the Expression for the Status Label**

- If you have many items to display or they change often, use a “Status Table,” which uses a data table to populate the statuses automatically. First, from the *Data tab*, insert a new table named **StatusLabelOutput**, as seen in Figure 9.44, where the **Material** column is a String Standard Property, and the **In Stock** column is an Expression Standard Property. Add a “Status Table” from the Animation tab specifying the **StatusLabelOutput** as the **Table Name** value.

Table Symbol	Table Label Output	
	Material	In Stock
1	SubA	MatSubA.QuantityInStock
2	SubB	MatSubB.QuantityInStock
3	Connectors	MatConnectors.QuantityInStock

Material	In Stock
SubA	166
SubB	166
Connectors	332

**Figure 9.44: Using a Status Table vs Individual Labels**

**Step 13:** Save and run the model for eight hours and answer the following questions.

**Question 24:** What are the average **SubAAvailable**, **SubBAvailable**, and **Connectors available**?

---

**Question 25:** How many kits were produced during the eight hours?

---

## Part 9.8: Commentary

- This is one of the most interesting chapters in that it illustrates how to model what would be considered a very complex operation composed of inventory concerns and supply chain issues. It's a powerful lesson, and the topic of inventory and supply will be considered in additional chapters, along with some of the SIMIO features illustrated here.
- **TASK SEQUENCES, MATERIALS, and BILL OF MATERIALS** is an extremely powerful modeling concept with numerous applications.

# Chapter 10

## Inventories, Supply Chains, and Optimization

Many industries/companies are facing many challenges in their effort to compete in the global marketplace. Demand variations, long lead times, and raw material supply fluctuations can result in excessive finished goods inventory and/or poor customer service levels. Many companies with longer lead times are forced into a make-to-stock policy to respond to customer demands and quickly maintain market share. Too much inventory will create higher costs and may lead to product obsolescence or spoilage, while too little inventory may cause stockouts and lower customer service levels. Therefore, designing and optimizing a company's supply chain has become a priority as it is becoming a necessity for the survival of many companies. The main objective of any supply chain is to supply the customer with a product when the customer wants it and at a price that maximizes profit. To achieve both objectives, a balance must be struck between carrying sufficient inventories to meet demand and not so much negatively impacting profitability.

### Part 10.1: Building a Simple Supply Chain

Figure 10.1 shows a very simple three-tier supply chain where customers arrive at a store, which could be a collection of stores to purchase a product. The stores place orders at the manufacturer's DC (Distribution Center), which will supply the product if it is in stock. The DC uses an inventory model to determine reorder policies for the supplier, which is assumed to have a raw material source. It takes between three to six days to truck the product from the supplier to the manufacturer.

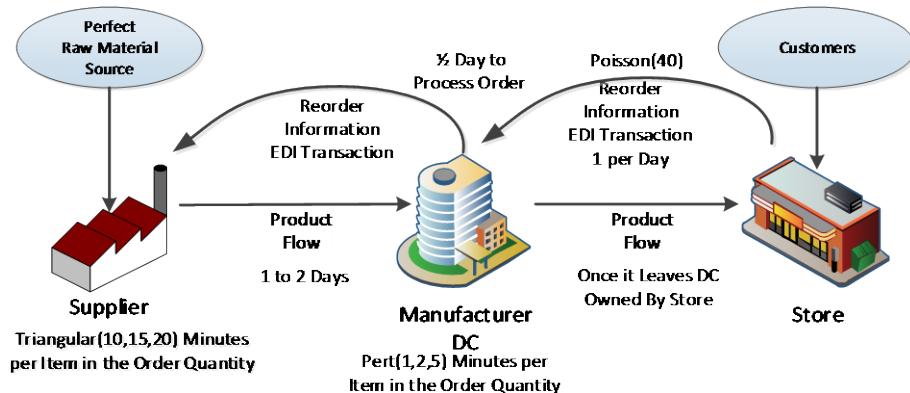
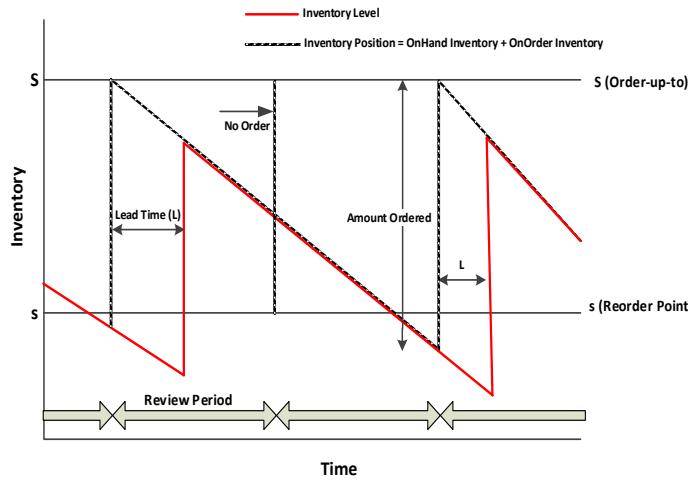


Figure 10.1: Simple Three-Tier Supply Chain

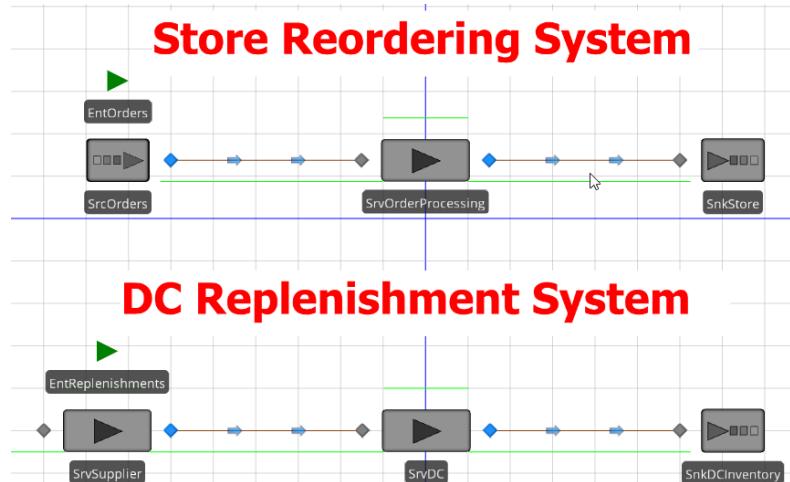
The goal is to track the DC's average inventory and service level/fill rate over a year. Therefore, the store customers will not be part of the current model since the manufacturer is interested in determining the reorder point and reorder quantity. The store reorders daily from the DC, and the DC utilizes a  $(s, S)$  periodic review inventory model, as seen in Figure 10.2. In this model, the inventory is reviewed at every fixed time interval, called the period. An amount  $s$  is the re-order point, a quantity below which triggers a re-order. The quantity  $S$  is determined as the quantity needed after inventory is re-ordered (an order-up-to amount).



**Figure 10.2: (s, S) Inventory Policy**

**Step 1:** Create a new model with two separate systems, as seen in Figure 10.3.

- The first system will model the ordering portion of the supply chain (i.e., orders from the store placed at the DC will need to be processed). Insert a SOURCE named **SrcOrders**, a SERVER named **SrvOrderProcessing**, and a SINK named **SnkStore**. A new MODELENTITY named **EntOrders** should also be inserted, making sure the **SrcOrders** creates these **EntOrders**. Use CONNECTORS to link the objects together, modeling Electronic Data Interchange transactions between the stores and the DC.
- The second system will model the flow of products from the back of the chain to the front of the supply chain used to replenish the inventory in the manufacturer's DC. Insert two SERVERS named **SrvSupplier** and **SrvDC**, a SINK named **SnkDCInventory**, and another MODELENTITY named **EntReplenishments**. Connect the **SrvSupplier** and **SrvDC** via a TIME PATH that uniformly takes between one and two days. Use a CONNECTOR from the **SrvDC** to the **SnkDCInventory** since the model is really only concerned with the DC.

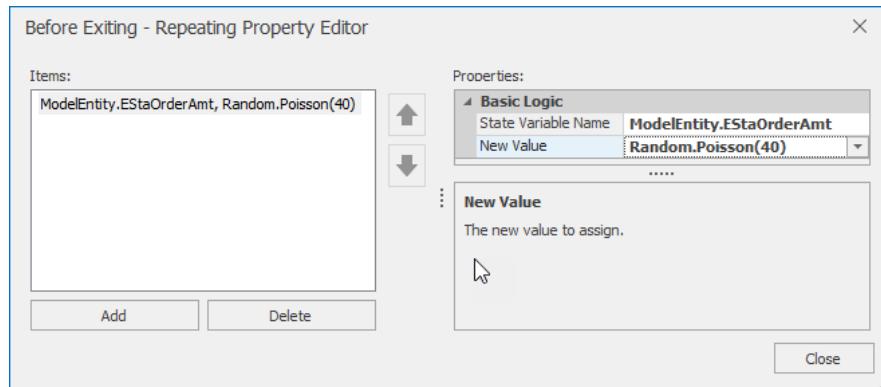


**Figure 10.3: SIMIO model of the Three Tier Supply Chain Re-ordering Process**

**Step 2:** Since we will not model every item in the inventory system as a separate entity, add a DISCRETE INTEGER STATE variable named **EStaOrderAmt** to the MODELENTITY<sup>131</sup> to represent the quantity of products for each store order or replenishment re-order.

**Step 3:** For the Ordering process, set the following information.

- Change the **SrcOrders** to have one order arrive daily.
- The amount of each order that arrives daily from the store is Poisson distributed with a mean of 40. As seen in Figure 10.4, use the *State Assignments→Before Exiting* property of the SOURCE to set the ModelEntity.EStaOrderAmt to a Random.Poisson(40).



**Figure 10.4: Specifying the Order Amount of each Order**

**Step 4:** Next, under the *Definitions→Properties* of the model, insert four *Expression Standard Properties* named **InitialInventory**, **ReorderPoint**, **OrderUptoQty**, and **ReviewPeriod**. These should all be placed in the “Inventory” Category.<sup>132</sup>

**Table 10.1: The Default Values for the Three Properties**

Property	Default Value	Category	Unit Type	Description
<b>InitialInventory</b>	700	Inventory	Unspecified	The initial inventory at the start of the simulation.
<b>ReorderPoint</b>	300	Inventory	Unspecified	Determines the point to re-order.
<b>OrderUptoQty</b>	700	Inventory	Unspecified	The maximum inventory to order up to quantity.
<b>ReviewPeriod</b>	5	Inventory	Time (Days)	The periodic review period

*Question 1:* What are the advantages of setting up properties for this model?

---

*Question 2:* How do you set the values for these properties of the Model?

---

**Step 5:** In the Model, insert a DISCRETE INTEGER STATE variable named **GStaInventory**, which will represent the current inventory amount (“OnHand”) of the product at the DC, and another one named **GStaOnOrder** (i.e., work in process), which represents the total amount the DC has currently ordered

---

<sup>131</sup> Select the MODELENTITY in the [Navigation] panel and go the *Definitions→States* section to add the integer state variable.

<sup>132</sup>. The first time, you will need to type the category name “Inventory,” which can then be selected from the dropdown box.

(“OnOrder”) from the supplier. The *Initial State Value* property should be 0 for both the **GStaInventory** and **GStaOnOrder** variables, respectively.

**Step 6:** Next, we need to insert several ELEMENTS into the model, as seen in Figure 10.5.

- Insert a **TALLY STATISTIC** named **TallyStatFR** to track the fill rate (i.e., service level performance).
- Insert a **STATESTATISTIC** named **StateStatInv** to track the inventory level performance. Specify the *State Variable Name* property as the **GStaInventory** variable.
- Insert a **TIMER** named **TimerReview** to model the review period. The DC currently reviews its inventory level every five days and should utilize the **ReviewPeriod** property (see Figure 10.5).

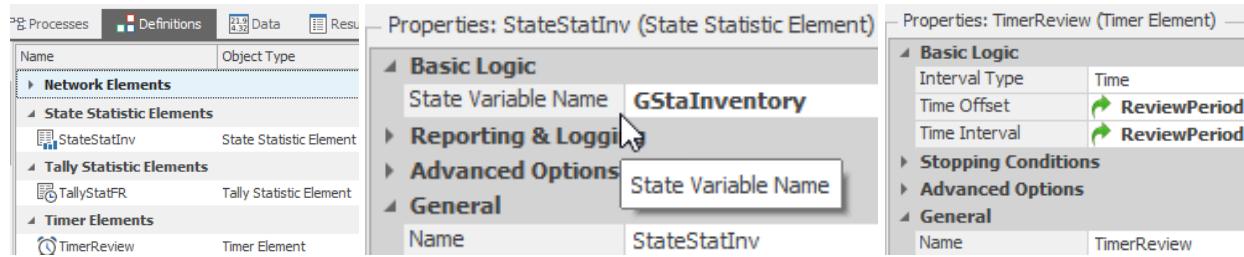


Figure 10.5: Setting up the Statistics and Periodic Review Timer

**Step 7:** Change the *Processing Time* property of **SrvOrderProcessing** such that orders take  $\frac{1}{2}$  a day to be processed before an attempt is made to fill the order with the available inventory.

**Step 8:** The supplier will take between 10 and 20 minutes, with the most likely time being 15 minutes, to process each product. Therefore, the total processing time for one order would be the sum of several triangular distributions (i.e., an order amount).<sup>133</sup> Again, the `Math.SumOfSamples` function handles the processing of the batch correctly by sampling from the distribution the appropriate number of times. For the **SrvSupplier**, specify the *Processing Time* property to `Math.SumOfSamples(Random.Triangular(10,15,20), ModelEntity.EstaOrderAmt)` uses the order amount to determine the number of triangular distributions to sample and sum, as seen in Figure 10.6.

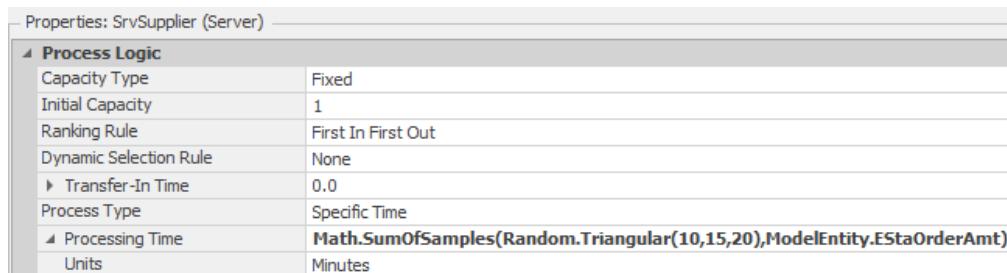


Figure 10.6: Specifying the Operation Quantity and Batch Size for Processing an Order

**Step 9:** Once the product arrives at the DC, the entire lot must be individually packaged before it is available to satisfy the store's demands. Therefore, set the SrvDC's Processing Time property to `Math.SumOfSamples(Random.Pert(1,2,5), ModelEntity.EstaOrderAmt)` in minutes.

<sup>133</sup> One may be inclined to use `ModelEntity.OrderAmt * Random.Triangular(10,20,30)` for the *Processing Time* property. However, this will not correctly model the summed distribution as seen in the example in Appendix A.

Properties: SrvDC (Server)	
Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Process Type	Specific Time
Processing Time	<code>Math.SumOfSamples(Random.Pert(1,2,5),ModelEntity.EStaOrderAmt)</code>
Units	Minutes

**Figure 10.7: Setting up the DC SERVER Properties to Process the Batch**

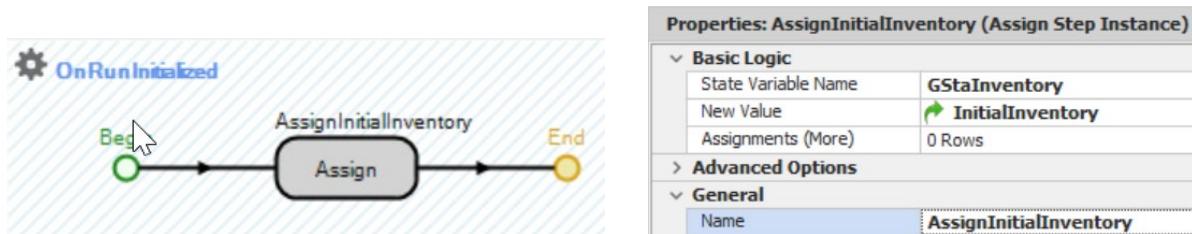
**Step 10:** Save and run the model for 26 weeks.

*Question 3:* How many store orders were produced during that period, and what is the average time in the system?

## Part 10.2: Processing Orders in the Supply Chain System

Recall that the initial inventory level property (i.e., **InitialInventory**) was created to allow for changing the initial condition at the model level.

**Step 1:** Since state variable default values cannot be reference properties, the state variable **GStaInventory** needs to be initialized at the beginning of the simulation via the *OnRunInitialized* process of the model. Insert this process by selecting it via the *Processes*→*Select Process*→*OnRunInitialized* dropdown menu. Insert an *Assign* step that assigns the **GStaInventory** state variable the **InitialInventory** value, as seen in Figure 10.8.



**Figure 10.8: Assigning the Initial Inventory Level**

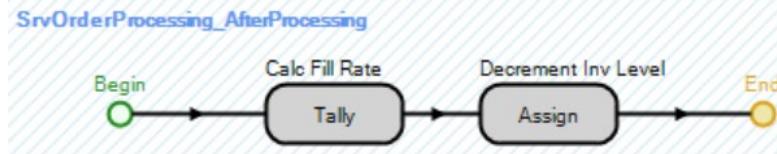
**Step 2:** **EntOrders** must be filled based on the current inventory position when they arrive at the DC and have been processed.<sup>134</sup> Therefore, insert a new “After Processing” add-on process trigger for the **SrvOrderProcessing** SERVER to handle the behavior (i.e., update the current inventory position and service level).

- Add a *Tally* step to update the TALLY STATISTIC **TallyStatFR** with the value equal to `Math.Min(1, GStaInventory/ModelEntity.EStaOrderAmt)`.<sup>135</sup>
- Next, insert an *Assign* step that will update the current **GStaInventory** level by subtracting the **EStaOrderAmt** of the particular order from the inventory on hand. However, if the **GStaInventory** level is less than the order amount, it would be negative. Since we are not allowing backorders, we should set the level to the following expression, as seen in Figure 10.10.

<sup>134</sup> Inventory position is defined as on-hand inventory plus on-order inventory (to account for orders that haven't arrived).

<sup>135</sup> Some may argue this service level calculation is really a fill rate, if the inventory is greater than the order amount then 100% of the demand appears to be satisfied.

- $\text{Math.Max}(0, \text{GStaInventory-ModelEntity.EStaOrderAmt})$



**Figure 10.9: Process to Fill Demands from the Stores**

Properties: CalcFillRate (Tally Step Instance)		Properties: Decrement Inv Level (Assign Step Instance)	
Basic Logic		Basic Logic	
Tally Statistic Name	TallyStatFR	State Variable Name	GStaInventory
Value Type	Expression	New Value	$\text{Math.max}(0, \text{GStaInventory-ModelEntity.EStaOrderAmt})$
Value Expression	$\text{Math.min}(1, \text{GStaInventory/ModelEntity.EStaOrderAmt})$	Assignments (More)	0 Rows
Tallies (More)	0 Rows		

**Figure 10.10: Properties of the Steps**

**Step 3:** To track the information while the simulation is running, insert a “STATUS TABLE” from the “Animation” tab, as seen in Figure 10.12, to track the current average inventory and service levels. To populate the “Status Table,” insert a *Data table* named **TableLabels** with a String Property column named **Type** and an *Expression Property* column named **Value**. The first column of labels is the description, while the second column uses the expressions `GStaInventory`, `GStaOnOrder`, `GStaInventory+GStaOnOrder`, `StateStatInv.Average`, and `TallyStatFR.Average`.<sup>136</sup>

Table Labels		
	Type	Value
1	Current Inventory	GStaInventory
2	Current On Order	GStaOnOrder
3	Inventory Position	GStaInventory + GStaOnOrder
4	Avg Inventory	StateStatInv.Average
5	Avg Fill Rate	TallyStatFR.Average

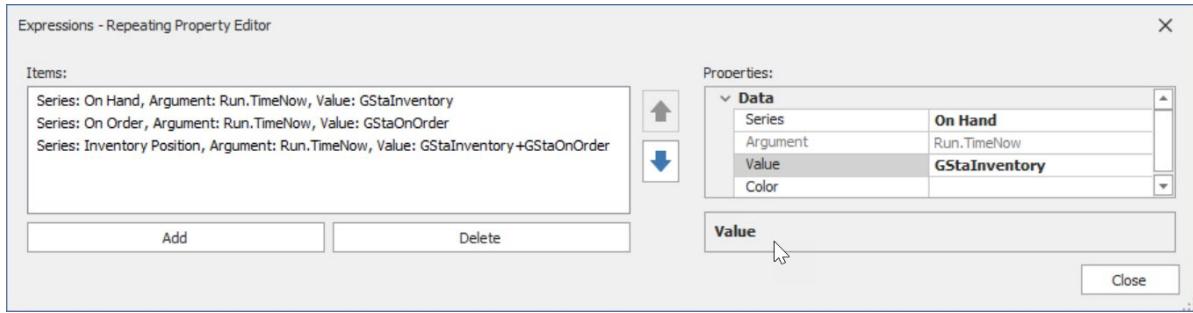
**Figure 10.11: Table to Be Used with a Status Table**

Type	Value
Current Inventory	27
Current On Order	0
Inventory Position	27
Avg Inventory	344.250000000006
Avg Fill Rate	1

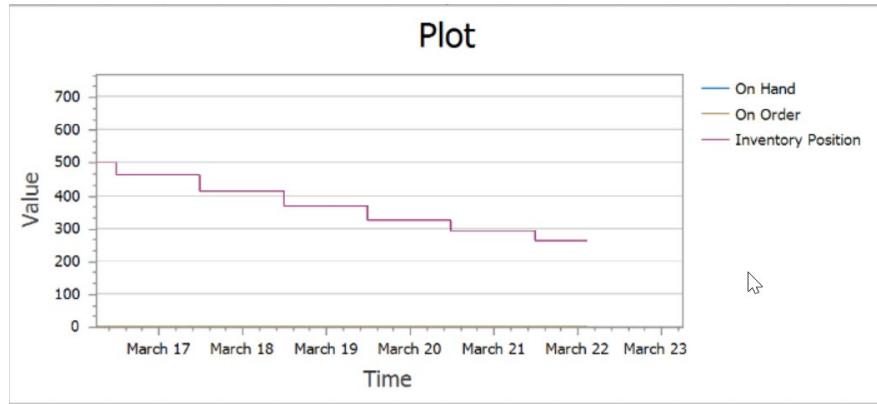
**Figure 10.12: Status Table to Show Current Inventory Level and Average Inventory and Fill Rate**

**Step 4:** From the “Animation” tab, insert a new STATUS PLOT that will track the current onhand inventory level, the replenishment on order, and the inventory position representing the onhand plus the on order. Using the *Additional Expression* property, add three items to the plot, as seen in Figure 10.13, and make the “Time Range” equal to “1” week.

<sup>136</sup> The advantage of using a status table rather than a series of status labels is that you only need to update the table to remove or add metrics to track which could be linked to a spreadsheet.



**Figure 10.13: Setting up the Status Plot to Track Inventory and On Order Amounts**



**Figure 10.14: Status Plot Showing Inventory Positions**

**Step 5:** Save and run the model (you may need to adjust the speed factor to 5000 or fast forward).

*Question 4:* What is the current inventory level?

---

*Question 5:* What is the average inventory level and service level?

---

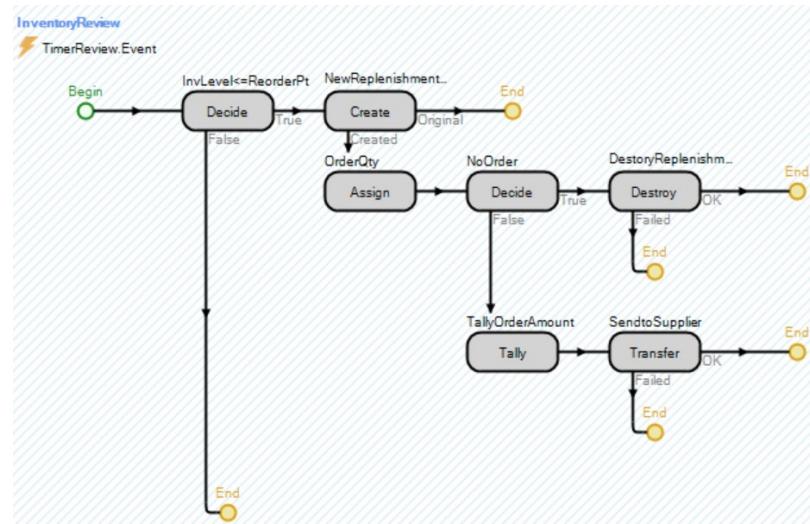
### Part 10.3: Creating the Replenishment Part of the Supply Chain System

The previous section handled demand orders from the stores, but once the inventory was depleted, there was no inventory replenishment. Hence, the service level was abysmal. The timer was set up to fire every five days. At that time, if the current inventory is below the re-order point, a replenishment order should be sent to the supplier based on the order-up-to quantity, the current inventory level, and the current number of outstanding orders (i.e., On Order). Recall that the “inventory position” is defined as the current inventory on hand plus the inventory represented in outstanding orders.

**Step 1:** Create some statistics on the replenishment procedure:

- Insert a STATE STATISTIC, called **StateStatOnOrder**, to keep track of the amount of inventory on order over time. This statistic should reference the **GStaOnOrder** state variable.
- Insert a TALLY STATISTIC, called **TallyStatAmtOrdered**, to record the amount ordered from the supplier.
- Add **StateStatOnOrder.Average** and **TallyStatAmtOrdered.Average** to the TableLabels to update the “Status Table.”

**Step 2:** Set up the periodic review to happen by creating a process to respond to the timer **TimerReview**. From the “Processes” tab, insert a new process named **InventoryReview**. Set the *Triggering Event* property to react to the **TimerReview.Event**, which causes the process to be executed every time the timer event fires, as seen in Figure 10.15.



**Figure 10.15: Process to Re-order**

- Insert a *Decide* step that uses a “ConditionBased” check to see if the current inventory is less than the re-order point (i.e., `GStaInventory <= ReorderPoint`).
- If a new replenishment (i.e., “True” branch) is needed, use the *Create*<sup>137</sup> step to create a new **EntReplenishments** entity, as seen in Figure 10.16.



**Figure 10.16: Creating a New ReOrder to Send the Supplier**

- For the newly created **EntReplenishments**, use an *Assign* step to assign the order amount and then increase the **StaOnOrder** value based on the order amount.

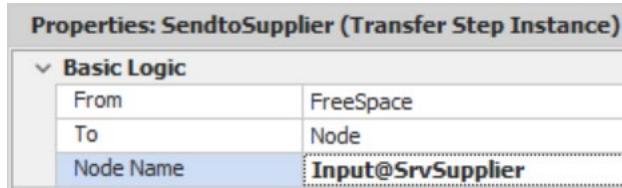
**Table 10.2: Determining the Order Amount and Updating the WIP**

State Variable	New Value
<b>ModelEntity.EStaOrderAmt</b>	<code>Math.Max(0, OrderUptoQty-GStaInventory-GStaOnOrder)</code>
<b>GStaOnOrder</b>	<code>GStaOnOrder + ModelEntity.EStaOrderAmt</code>

- In the second *Decide* step, determine if the **EStaOrderAmt** is zero (i.e., `ModelEntity.EStaOrderAmt==0`). Namely, no replenishment order is needed (an assumption here is that there are never any zero quantity replenishment orders)
- If no replenishment order is needed, then the replenishment order entity is destroyed using the *Destroy* step.

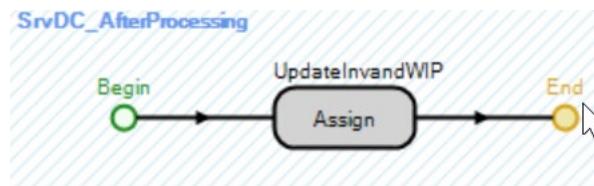
<sup>137</sup> The *Create* step can create new objects or copies of the associated or parent objects. Each of the created objects will be executed by their own token. Note that created tokens are updated in SIMIO before the original.

- Otherwise, the amount of the replenishment order (`ModelEntity.EStaOrderAmt`) is tallied via the **TallyStatAmtOrdered**.
- Once the **EntReplenishment** has been tallied, it must be sent to the supplier using a *Transfer* step to move it from “FreeSpace” to the **Input** node of the **WrkSupplier**.<sup>138</sup>



**Figure 10.17: Sending the New Re-order to the Supplier**

**Step 3:** When the products arrive back at the DC and have been packaged, the **GStaInventory** variable must be increased by the order amount. At the same time, the **same value reduces the GStaOnOrder variable**. Insert the “*AfterProcessing*” add-on process trigger for the **SrvDC**, as seen in Figure 10.18. Insert an *Assign* step that updates the values in Table 10.3.



**Figure 10.18: Updating the Inventory and WIP Values**

**Table 10.3: Updating Inventory and DC once Product Arrives**

State Variable	New Value
<code>GStaInventory</code>	<code>GStaInventory + ModelEntity.EStaOrderAmt</code>
<code>GStaOnOrder</code>	<code>GStaOnOrder - ModelEntity.EStaOrderAmt</code>

**Step 4:** Save and run the model (you may need to fast-forward the simulation) for 26 weeks.

*Question 6:* What is the average inventory level, the inventory on order, and the service level?

---

*Question 7:* What is the average amount ordered?

---

*Question 8:* Is this an adequate system?

---

**Step 5:** You may notice that the status table displays as many significant digits as possible for the average inventory and the service level/fill rate. SIMIO exposes many of the .Net string manipulation functions. For example, the `String.Format` function can be used to format the output.<sup>139</sup> Replace the two average and service-level status labels with the ones in Table 10.4.

---

<sup>138</sup> When entities are created, they are placed in “FreeSpace” and, typically, will need to be “transferred” from there to one of the model nodes or stations.

<sup>139</sup> The syntax for the format function is `String.Format(string, arg1, arg2, ...)` where the format “string” can contain place holders that are enclosed by {} which start with the 0<sup>th</sup> argument (i.e., {0}). For example, `String.Format("SL = {0} and the Avg = {1}", 0.45, 89.4)` has two arguments {0} and {1} and replaces those with the numbers 0.45 and 89.4 respectively to produce a ‘SL=0.45 and the Avg = 89.4’. Along with the arguments, one can specify formatting information especially as applied to

**Table 10.4: Formatting the Output of the Status Labels**

Status Label	Expression
Average Inventory	string.format("{0:0.##}", StateStatInv.Average)
Average OnOrder	string.format("{0:0.##}", StateStatOnOrder.Average)
Average Fill Rate	string.format("{0:0.00%}", TallyStatFR.Average )
Average Replenishment Qty	string.format("{0:0.##}", TallyStatAmtOrdered.Average)

**Step 6:** Save and run the model (you may need to adjust the speed factor to 100 or fast forward the simulation).

**Question 9:** Were the average inventory, on-order, fill rate, and replenishment quantity levels now formatted correctly?

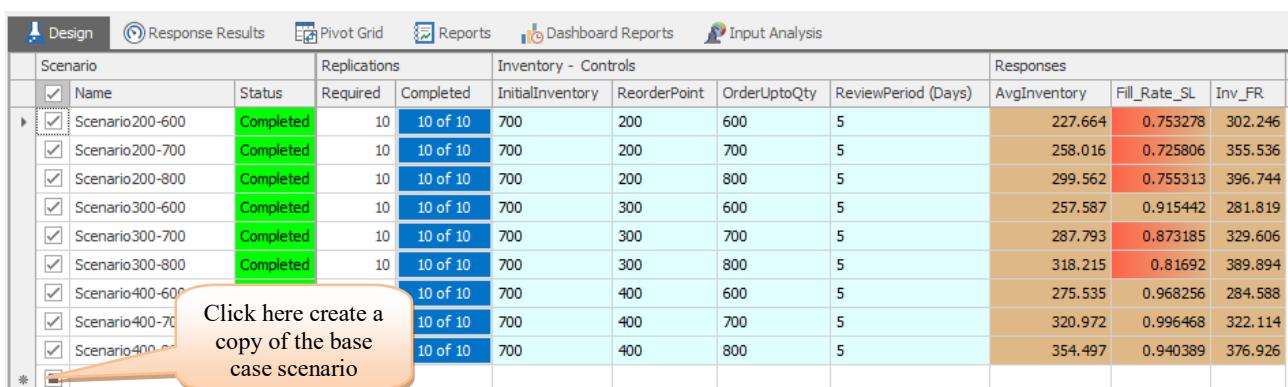
---

### Part 10.4: Using an Experiment to Determine the Best Values

Simulation is often used to improve a system. Can we find a better order-up-to-quantity and re-order point that improves the system's overall performance? Can SIMIO provide help with this kind of question? Or can SIMIO help in searching for an improved system? The answer is yes to all these questions.

**Step 1:** From the *Project Home*→*Create* section, insert a new “Experiment” named **FirstExperiment**. This will create a new experiment window and automatically create the first scenario, which is the base model, as seen in Figure 10.19. SIMIO automatically added the four properties as control variables.<sup>140</sup>

**Step 2:** Insert eight more scenarios by clicking the little box in the last row. This will copy all of the parameters from the base case. Then, change re-order points and order up to quantities to do a full factorial experiment of the three re-order points (200, 300, and 400) and order up to quantity (600, 700, 800), as seen in Figure 10.19. Change the names of the scenarios so they can be interpreted.



Scenario		Replications		Inventory - Controls					Responses		
	Name	Required	Completed	InitialInventory	ReorderPoint	OrderUptoQty	ReviewPeriod (Days)	AvgInventory	Fill_Rate_SL	Inv_FR	
✓	Scenario200-600	Completed	10	10 of 10	700	200	600	5	227.664	0.753278	302.246
✓	Scenario200-700	Completed	10	10 of 10	700	200	700	5	258.016	0.725806	355.536
✓	Scenario200-800	Completed	10	10 of 10	700	200	800	5	299.562	0.755313	396.744
✓	Scenario300-600	Completed	10	10 of 10	700	300	600	5	257.587	0.915442	281.819
✓	Scenario300-700	Completed	10	10 of 10	700	300	700	5	287.793	0.873185	329.606
✓	Scenario300-800	Completed	10	10 of 10	700	300	800	5	318.215	0.81692	389.894
✓	Scenario400-600		10	10 of 10	700	400	600	5	275.535	0.968256	284.588
✓	Scenario400-700		10	10 of 10	700	400	700	5	320.972	0.996468	322.114
✓	Scenario400-800		10	10 of 10	700	400	800	5	354.497	0.940389	376.926

**Figure 10.19: First Experiment Trying Different Re-order Points and Order Up to Quantities**

**Step 3:** Using the *Design*→*Experiment*→*Add Response* button, insert three responses that will be used (see Figure 10.19) to select the best scenario with the following parameters specified in Table 10.5. The first

numbers by using a colon on the argument then specifying the number format. For example, {0:#.##} will display only two significant digits while {0:#.00%} will display a percentage and force there to always be two significant digits.

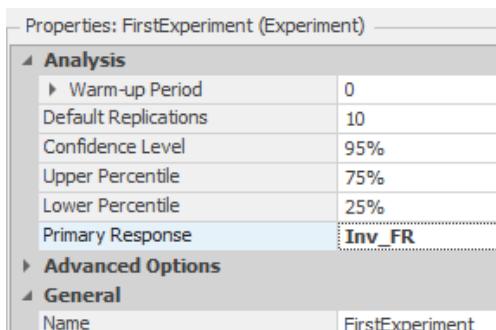
<sup>140</sup> Other control variables can be defined.

two responses seem very logical. In this model, these responses are typically conflicting (i.e., as the inventory level is minimized, the service level goes down). Therefore, the third one converts the multi-objective problem into a single response. This new response goes down as the service level increases and the inventory decreases. The lower and upper bounds on the responses will be used to flag responses outside a stipulated range.<sup>141</sup> In this case, we would like to have a service level greater than 90%.

**Table 10.5: Experimental Responses**

Name	Expression	Objective	Lower Bound
AvgInventor_y	StateStatInv.Average	Minimize	
Fill_Rate_DL	TallyStatFR.Average	Maximize	0.90
Inv_FR	StateStatInv.Average/TallyStatFR.Average	Minimize	

**Step 4:** Figure 10.20 shows the basic experiment parameters, where you can define the warm-up period and the default number of replications that all scenarios will run. For the SMORE plots and optimization, the confidence interval and upper and lower percentiles are displayed. Also, you need to define the primary response that the ranking and selection will use and Optquest™ add-ins (see the next sections).



**Figure 10.20: Setting up the Experiment Parameters**

**Step 5:** Save the model and run the experiment.

**Question 10:** Based on the responses, which re-order point and order up to quantity seems best?

---

**Question 11:** How did you make this determination?

---

## Part 10.5: Using SMORE Plots to Determine the Best Values

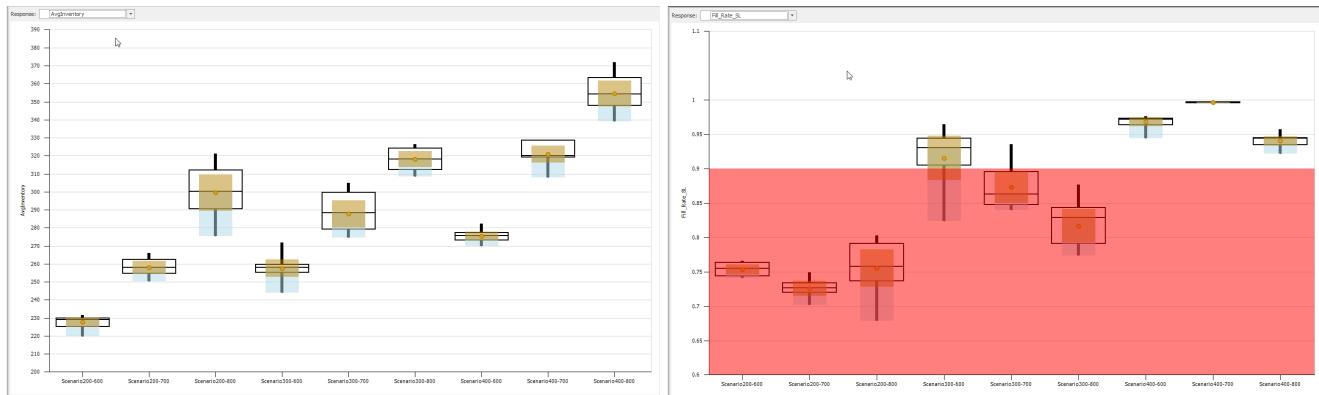
In the previous section, we just used the average of the ten replications to assist you in choosing the best selection. The difficulty is that the variability of the system is not accounted for when selecting the best scenario. “SIMIO Measure of Risk & Error” (SMORE) plots offer the ability to see the variability and

---

<sup>141</sup> The values in the response that don't fall within the bounds are shown in a red gradient color.

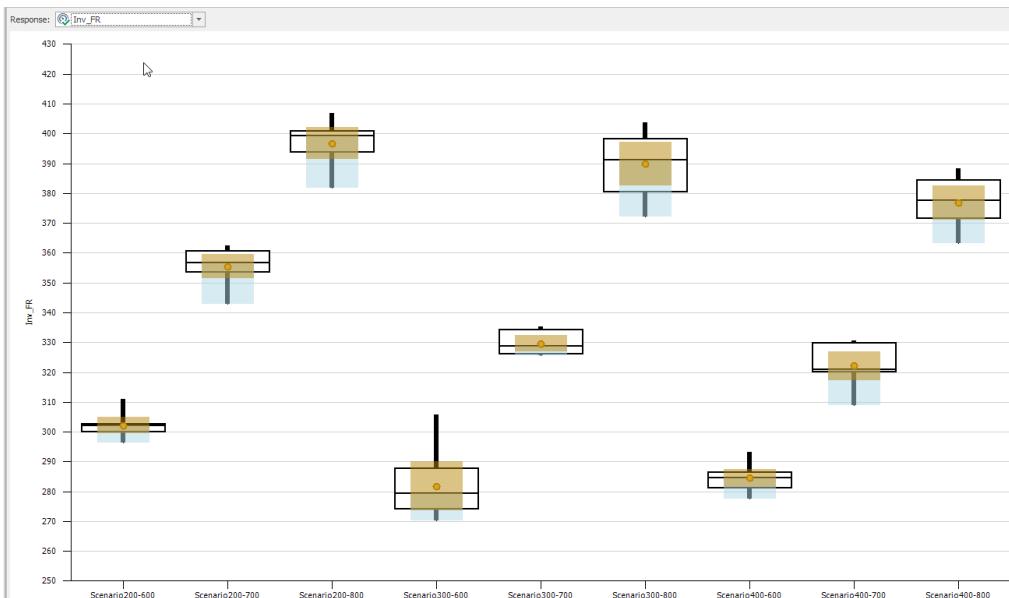
confidence intervals. They show the significant percentiles (median or 50<sup>th</sup> percentile, lower chosen percentile, and upper chosen percentile), mean, and confidence intervals for the mean and the percentiles.

**Step 1:** Select the “Response Results” tab from the previously run model to see the SMORE plots. The primary response (Inv\_FR) will be displayed by default, but the other response can be selected from the dropdown, as seen in Figure 10.21. Quickly, we can see from the “AvgInventory” response that Scenario200-600 seems to be the best, and its variation is clear from the other plots. Scenario 300-600 appears to be the second best; however, its variation overlaps with Scenario 200-700. However, when looking at the service level with the limits turned on, only Scenario300-600, Scenario400-600, Scenario400-700, and Scenario400-800 meet the 90% service level limit, with the Scenario400-700 seeming statistically the best. Considering the two plots together, Scenerio300-600 appears to be the better choice.



**Figure 10.21: SMORE Plots of the Average Inventory and Average Service Level**

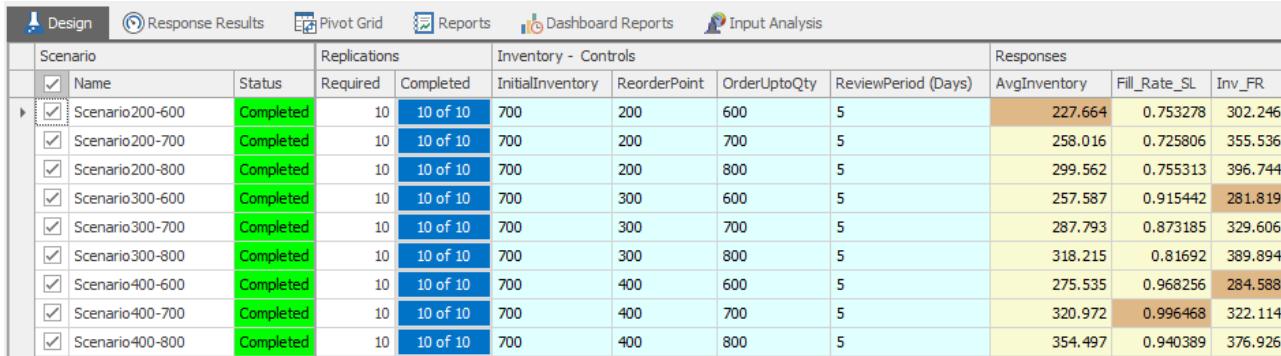
**Step 2:** Looking at the Primary Response (“Inv\_FR”) SMORE Plot in Figure 10.22, Scenario 300-600 appears to be the best, based on the combined response. However, its confidence interval does overlap with Scenario400-600.<sup>142</sup>



**Figure 10.22: SMORE Plots of the Combined Inventory/Service Level Response**

<sup>142</sup> Refer back to Chapter 8 for more explanation of SMORE plots and their meanings.

**Step 3:** By clicking the “Subset Selection” option in the “Analysis” section of the ribbon in the “Design” tab, you can see the choices made by a series of algorithms within SIMIO. Scenarios for each response are divided into the “best possible group” and the “rejects group”, as shown in Figure 10.23.



Scenario		Replications		Inventory - Controls					Responses		
	Name	Status	Required	Completed	InitialInventory	ReorderPoint	OrderUptoQty	ReviewPeriod (Days)	AvgInventory	Fill_Rate_SL	Inv_FR
▶	Scenario200-600	Completed	10	10 of 10	700	200	600	5	227.664	0.753278	302.246
	Scenario200-700	Completed	10	10 of 10	700	200	700	5	258.016	0.725806	355.536
	Scenario200-800	Completed	10	10 of 10	700	200	800	5	299.562	0.755313	396.744
	Scenario300-600	Completed	10	10 of 10	700	300	600	5	257.587	0.915442	281.819
	Scenario300-700	Completed	10	10 of 10	700	300	700	5	287.793	0.873185	329.606
	Scenario300-800	Completed	10	10 of 10	700	300	800	5	318.215	0.81692	389.894
	Scenario400-600	Completed	10	10 of 10	700	400	600	5	275.535	0.968256	284.588
	Scenario400-700	Completed	10	10 of 10	700	400	700	5	320.972	0.996468	322.114
	Scenario400-800	Completed	10	10 of 10	700	400	800	5	354.497	0.940389	376.926

**Figure 10.23: Using Subset Selection**

While the best possible group may consist of scenarios that cannot be proven statistically different from each other, it can be shown that the best possible group scenarios will be statistically better than the rejected group (shown in muted color). You can see that the two best scenarios on response Inv\_FR are not statistically different.

**Question 12:** Which scenario(s) is(are) statistically selected as the best for the three responses?

---

## Part 10.6: Using Ranking and Selection to Determine the Best Scenario

The analysis is quite simple (focused on only two variables within the model), and ten replications seemed enough to choose the best. However, using the SMORE plots still requires judgment or visual identification. Ranking and selection methods allow you to determine statistically which scenario is the best. One of the advantages of SIMIO is the two built-in state-of-the-art ranking and selection methods. One is based on research by Kim and Nelson (called “KN”),<sup>143</sup> and the other one is by Ma and Henderson (called GSP).<sup>144</sup> The GSP method may improve performance when doing large-scale models that require many scenarios in a parallel processing environment.

**Step 1:** Save the current model and copy the **FirstExperiment** by right-clicking it in the [Navigation] panel and choosing the *Duplicate Experiment* menu item. Name the new experiment **KNEperiment**.

**Step 2:** Next, choose “Select Best Scenario using KN” from the *Design→Add-Ins* section, as shown in Figure 10.24.

---

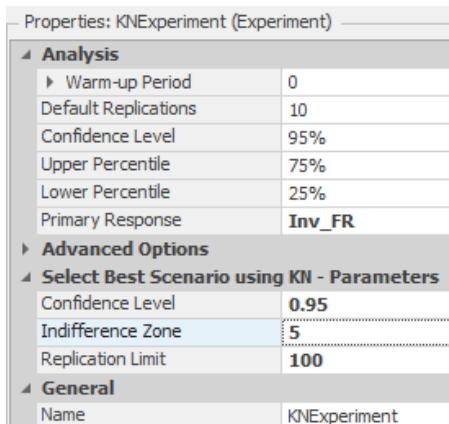
<sup>143</sup> S. Kim and B. L. Nelson, "A Fully Sequential Procedure for Indifference-Zone Selection in Simulation," *ACM Transactions on Modeling and Computer Simulation* 11 (2001), 251-273.

<sup>144</sup> Sijia Ma, Shane G. Henderson, "Predicting the Simulation Budget in Ranking and Selection Procedures," *ACM Transactions on Modeling and Computer Simulation* 29(3): 14:1-14:25 (2019)



**Figure 10.24: Selecting the KN Algorithm Add-in**

**Step 3:** This will add the KN algorithm to the experiment (see Figure 10.23) and give you additional properties. Set the Indifference Zone property to five, which stipulates that one solution can be determined better than another only if the two solutions differ by more than five. Since the ranking and selection scheme will change the number of replications during its execution, the *Replication Limit* property specifies the maximum number that can be run.



**Figure 10.25: Setting up the KN Parameters**

**Step 4:** Set the number of “Required” replications (under the Replications category) to one each, make sure that each scenario is “checked” (in the left column), and “Reset” the experiment.

**Step 5:** Run the experiment and notice that the number of replications required for each scenario increases to ten. The KN algorithm requires at least ten replications to detect a difference among scenarios reliably.

*Question 13:* Based on the differences among the scenarios **Inv\_FR**, do you think a larger number of replications is needed to determine differences that are more significant than five?

---

*Question 14:* What does running more replications do for the analysis?

---

**Step 6:** The KN algorithm has selected Scenario 300-600 as the best (the only one checked), as seen in Figure 10.26. Figure 1.23 shows that Scenario 300-600 and Scenario 400-600 were statistically identical. KN required additional replications on those two scenarios to determine which was statistically better on the indifference zone of five.

Scenario			Replications		Inventory - Controls				Responses		
	Name	Status	Required	Completed	InitialInventory	ReorderPoint	OrderUptoQty	ReviewPeriod (Days)	AvgInventory	Fill_Rate_SL	Inv_FR
▶	Scenario200-600	Completed	10	10 of 10	700	200	600	5	227.664	0.753278	302.246
	Scenario200-700	Completed	10	10 of 10	700	200	700	5	258.016	0.725806	355.536
	Scenario200-800	Completed	10	10 of 10	700	200	800	5	299.562	0.755313	396.744
	Scenario300-600	Completed	22	22 of 22	700	300	600	5	256.397	0.923154	278.106
	Scenario300-700	Completed	10	10 of 10	700	300	700	5	287.793	0.873185	329.606
	Scenario300-800	Completed	10	10 of 10	700	300	800	5	318.215	0.81692	389.894
	Scenario400-600	Completed	22	22 of 22	700	400	600	5	274.094	0.969275	282.788
	Scenario400-700	Completed	10	10 of 10	700	400	700	5	320.972	0.996468	322.114
	Scenario400-800	Completed	10	10 of 10	700	400	800	5	354.497	0.940389	376.926

Figure 10.26: Results of running the KN

**Step 7:** Verify that Scenario 300-600 is selected and examine its SMORE plot in the “Response Chart.”

*Question 15:* What is the 95% confidence interval on the mean?

---

**Step 8:** Reset the experiment, select all the scenarios again, and change the required to 10. Change the indifference zone to one and rerun the experiment to see the impact. Figure 10.27 displays the algorithm's results. You can see two additional scenarios needed to be run for more than the default ten replications, and the two very close scenarios had to be run 86 times to distinguish them with an indifference zone of “1.”

Scenario			Replications		Inventory - Controls				Responses		
	Name	Status	Required	Completed	InitialInventory	ReorderPoint	OrderUptoQty	ReviewPeriod (Days)	AvgInventory	Fill_Rate_SL	Inv_FR
▶	Scenario200-600	Completed	12	12 of 12	700	200	600	5	227.434	0.753777	301.737
	Scenario200-700	Completed	10	10 of 10	700	200	700	5	258.016	0.725806	355.536
	Scenario200-800	Completed	10	10 of 10	700	200	800	5	299.562	0.755313	396.744
	Scenario300-600	Completed	86	86 of 86	700	300	600	5	256.814	0.93302	275.479
	Scenario300-700	Completed	10	10 of 10	700	300	700	5	287.793	0.873185	329.606
	Scenario300-800	Completed	10	10 of 10	700	300	800	5	318.215	0.81692	389.894
	Scenario400-600	Completed	86	86 of 86	700	400	600	5	273.333	0.970483	281.644
	Scenario400-700	Completed	14	14 of 14	700	400	700	5	321.257	0.99739	322.101
	Scenario400-800	Completed	10	10 of 10	700	400	800	5	354.497	0.940389	376.926

Figure 10.27: Running the KN Algorithm with an Indifference Zone of “1”

*Question 16:* Did KN need more than ten replications to distinguish the best scenario, and which scenario did it select this time?

---

## Part 10.7: Using OptQuest™ to Optimize the Parameters

The KN algorithm used in the previous section determined the optimal scenario from a fixed set of user-defined scenarios. SIMIO has an add-in optimizer, OptQuest™, that can perform traditional optimization by trying other values for the decision variables not specified. It will also allow constraints on the decision variables and output responses, which the KN and GSP cannot.<sup>145</sup>

**Step 1:** Save the current model and, duplicate the **FirstExperiment** and then rename the new experiment **OptQuestExperiment**.

**Step 2:** Use the “*Clear*” button to remove the Select Best Scenario add-in and then select the OptQuest for SIMIO add-in if you duplicated the **KNExperiment** instead.

---

<sup>145</sup> OptQuest is an additional add-in that has to be purchased.

**Step 3:** Since the optimizer will try different values for our control (decision) variables, we must define their valid ranges. Select appropriate ranges and increment values to speed up the convergence and potentially the solution quality. Select each control variable and set their values according to Figure 10.28. As you can see, the **Initial Inventory** is not included in the optimization. An *increment* value of 25 was chosen for the other two variables to allow the algorithm to select values only in increments of 25.<sup>146</sup>

Properties: InitialInventory (Control)	
▶ <b>InitialInventory</b>	
◀ <b>OptQuest for Simio - Parameters</b>	
Include in Optimization	No
Minimum Value	1
Maximum Value	5
Increment	1

Properties: ReorderPoint (Control)	
▶ <b>ReorderPoint</b>	
◀ <b>OptQuest for Simio - Parameters</b>	
Include in Optimization	Yes
Minimum Value	200
Maximum Value	500
Increment	25

Properties: OrderUptoQty (Control)	
▶ <b>OrderUptoQty</b>	
◀ <b>OptQuest for Simio - Parameters</b>	
Include in Optimization	Yes
Minimum Value	500
Maximum Value	1200
Increment	25

Properties: ReviewPeriod (Control)	
▶ <b>ReviewPeriod</b>	
◀ <b>OptQuest for Simio - Parameters</b>	
Include in Optimization	Yes
Minimum Value	1
Maximum Value	7
Increment	1

**Figure 10.28: Setting the Lower and Upper Bounds on the Decision Variables**

**Step 4:** Next, choose the “OptQuest™ for SIMIO” from the *Design→Add-Ins* section to insert the optimization algorithm. It will add a setup section in the **EXPERIMENT** properties, as seen in Figure 10.29. The algorithm uses a confidence level to distinguish between solutions. The relative error percentage of the confidence level is expressed as a percent of the mean. Notice that a range of replications is specified, and a maximum number of scenarios is specified. These will limit the time OptQuest will spend computationally in the simulations and search for a better solution.

---

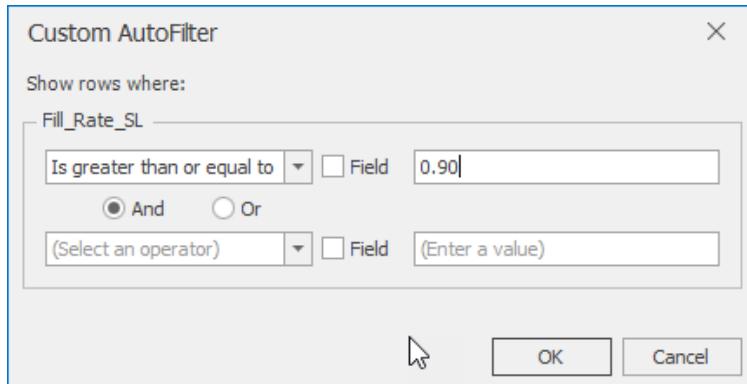
<sup>146</sup> Finer resolutions (i.e., smaller values) may lead to better solutions. However, this can affect the solution quality of the optimization algorithm. Having a coarse resolution quickly allows the algorithm to find a good region. A second optimization with a finer resolution could be run around the point found by the first optimization problem.

Properties: OptQuestExperiment (Experiment)	
<b>Analysis</b>	
Warm-up Period	0
Default Replications	10
Confidence Level	95%
Upper Percentile	75%
Lower Percentile	25%
Primary Response	<b>Inv_FR</b>
<b>Advanced Options</b>	
<b>OptQuest for Simio - Parameters</b>	
Min Replications	5
Max Replications	<b>20</b>
Max Scenarios	300
Confidence Level	95%
Relative Error	0.1
Objective Type	<b>Single Objective</b>

**Figure 10.29: OptQuest Parameters**

**Step 5:** Save and run the optimization algorithm. Notice how it tries several different variable values of the controls. As specified in the Max Scenarios property, it will create a maximum of 100 scenarios or decision points.

**Step 6:** Notice only feasible scenarios with respect to constraints on the responses (i.e., service level greater than 90%) are checked. Click on the little funnel ( in the upper right corner of the Fill\_Rate\_SL response and choose the [Custom] filter option. Set up the filter to only allow feasible scenarios to be visible, as seen in Figure 10.30. Then, sort the scenarios by the Inv\_FR column in ascending order to see those with minimum Inv\_FR ratios by right-clicking on the column heading.



**Figure 10.30: Filtering only Solutions that meet the Threshold of 90% Service Level**

**Question 17:** What appears to be the best re-order point and order up to quantity values considering not only the Inv\_FR ratio but the service level?

**Step 7:** Since the OptQuest™ only used five replications, variability may be a problem. Since the optimization parameters (i.e., bounds on the control variables will be lost), first, duplicate the experiment and name it OptQuestExperiment2 by right-clicking on the name in the [Navigation] panel.

**Step 8:** Select the **OptQuestExperiment** from the [Navigation] panel, clear the OptQuest add-in, add the Select Best Scenario, and make sure the Primary Response is **Inv\_FR**. Keep the top 10 (or some other

number) scenarios and delete the rest. Use an *Indifference Zone* property of “1” and rerun the KN algorithm.

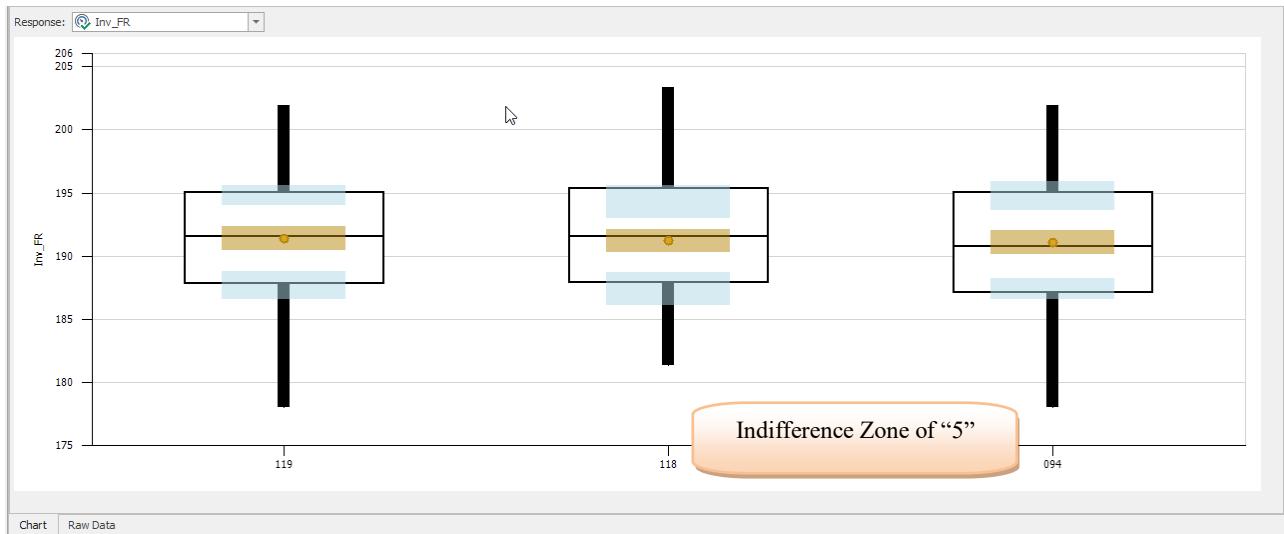
*Question 18:* What did you get for the best Re-order Point and Order Up to Quantity?

---

*Question 19:* What alternative do you choose if you want the service level to be at least 90%?

---

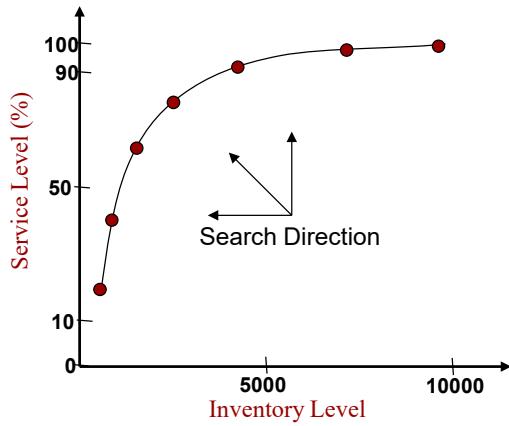
**Step 9:** To see that one is better than the others, examine the *Response Chart* for the one scenario KN selected, as seen in Figure 10.28. Note that you must “check” the alternative scenarios to see them in the Response Chart. In our example, Scenario 052 (i.e., 300 re-order points and 550 (order-up-to quantity)) was deemed to be the best, with one as the indifference zone.



**Figure 10.31: Viewing the Best Scenarios Indifference Zone of “5”**

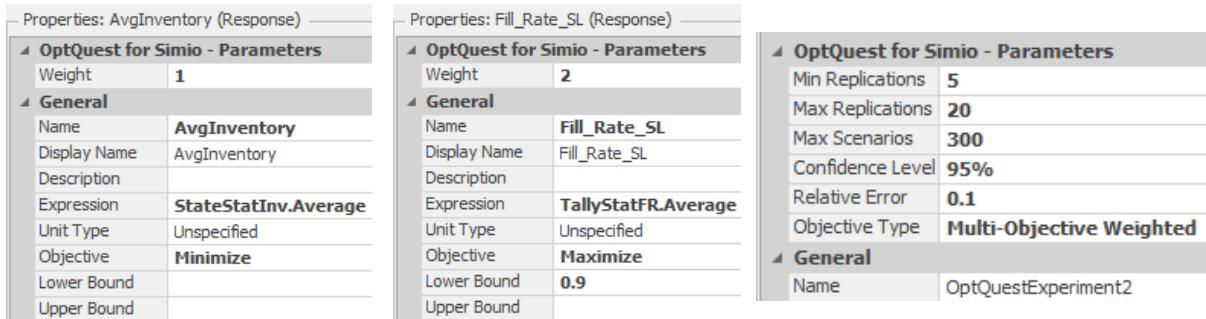
### Part 10.8: Multi-objective and Additional Constraints using OptQuest™

While the KN selection algorithm can statistically determine which scenario is the best, it cannot consider constraints on the responses. Constraints on the control variables should be handled since the scenarios selected should be feasible. Unfortunately, many problems often have multiple conflicting objectives, and it is rare for a single point to optimize all the objectives. Figure 10.32 depicts the efficiency frontier of a problem that optimizes both service level and inventory level at the same time. The frontier represents all non-dominated points that a decision-maker could choose. From the figure, the inventory level could be reduced by half if they were willing to go from 98% service level to around 90% service level.



**Figure 10.32: Efficiency Frontier**

Finding the frontier is very difficult. To help alleviate the problem, many people try to convert the multi-objective problem into a single objective ( $g(x)$ ) problem. The most common way and easiest way is to aggregate the objectives into a single objective by summing up the weighted objectives (e.g.,  $g(x) = \sum_{i=1}^k w_i f_i(x)$ ). The decision maker can determine the importance of each objective by choosing the weights accordingly. SIMIO can automatically create the combined function by specifying the appropriate weights of the responses and choosing the Multi-Objective Weighted value as the Objective Type property of the Experiment, as seen in Figure 10.33.



**Figure 10.33: Using the Multi-Objective Aggregate Optimization Method**

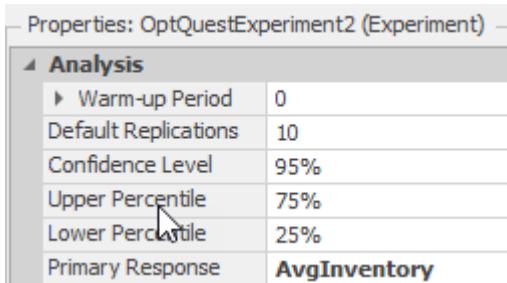
In this example, the decision maker puts twice as much importance on service level as on inventory. The difficulty with this multi-objective approach is scaling the weights to find solutions. In this example, inventory is in the hundreds while service level is less than one. In many cases, the optimization algorithm will only optimize the dominant objective. While this may be a convenient method, one can accomplish the same by creating an aggregate response and specifying the expression as `StateStatInv.Average + 2 * TallyStatSL.Average`. To avoid the scaling problem (i.e., 100s vs percentages), we created an aggregate response that was minimized by taking the inventory level divided by the service level to try to force the algorithm to minimize the inventory while maximizing the service level.

The other way is to optimize one primary objective (e.g., average inventory) while setting thresholds (i.e., lower and upper bounds) on the other objectives (e.g., the lower bound of 0.90 on service level), which would create one point on the frontier which we did in the previous section. There are two categories of constraints in the OptQuest for SIMIO: constraints on the controls (inputs) and constraints on the responses (outputs).

Suppose we consider a slightly different optimization problem, namely minimizing the average inventory level while keeping the service level above 95%. We can write this more formally using the following equations. Of the three constraints, the last two are simply bounds on the controls. The first constraint restricts the simulation's response (i.e., Fill Rate).

**Minimize:** AvgInventory  
**Subject to:** Fill Rate  $\geq .95$   
 $200 \leq \text{ReorderPoint} \leq 500$   
 $500 \leq \text{Order UpToQty} \leq 1200$

**Step 1:** Select the **OptQuestExperiment2** and change the primary objective to be the **AvgInventory** response and *Objective Type* to “Single Objective” under the experiment properties.<sup>147</sup>



**Figure 10.34: Change the Primary Response of the Experiment Properties**

**Step 2:** Save and run the optimization algorithm and compare the results to the previous experiment.

**Step 3:** Sort the scenarios in the average inventory column in ascending order to see those with minimum average inventory, and filter the service levels to display only feasible solutions.

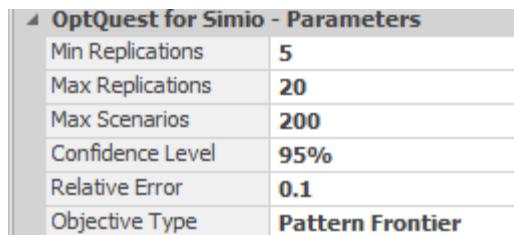
*Question 20:* What appears to be the best re-order point and order up to quantity values, considering the average inventory and service level?

---

**Step 4:** All of the previous methods made the decision before the search (i.e., the modeler decided which point on the Pareto frontier to generate). OptQuest can also discover the efficiency frontier. Duplicate the **OptQuestExperiment2** and name it **OptQuestExperiment3**.

**Step 5:** Remove the **Inv\_FR** response by selecting it and clicking the *Remove Response* button. SIMIO will generate the frontier based on all responses present.<sup>148</sup>

**Step 6:** Change the *Objective Type* property under the **OptQuestExperiment3** properties to “*Pattern Frontier*” and the *Max Scenarios* to “200,” as seen in Figure 10.35.



**Figure 10.35: Changing the Objective Type to Pattern Frontier**

**Step 7:** Select the **Service\_Level** response and change the lower bound to 0.5. This will generate points above the 50% service level.

---

<sup>147</sup> Right click the experiment name in the [Navigation] window to access the properties of the experiment.

<sup>148</sup> The frontier for two responses is a line and difficult to generate while three would be a plane needing a large number of scenarios.

Properties: Fill_Rate_SL (Response)	
OptQuest for Simio - Parameters	
General	
Name	Fill_Rate_SL
Display Name	Fill_Rate_SL
Description	
Expression	TallyStatFR.Average
Unit Type	Unspecified
Objective	Maximize
Lower Bound	0.5
Upper Bound	

Figure 10.36: Change the Lower Bound on the Service Level

**Step 8:** Save and run the model. After the model runs, sort the checked column in descending order to show all the points created on the frontier, as seen in Figure 10.34. Figure 10.38 shows the frontier graphed by copying the data to Excel. The data in Excel has had its duplicates removed and been sorted in descending order.

	Name	Status	Required	Completed	InitialInventory	ReorderPoint	OrderUptoQty	ReviewPeriod (Days)	AvgInventory	Fill_Rate_SL
✓	274	Completed	5	5 of 5	700	325	500	7	185.991	0.952778
✓	199	Completed	5	5 of 5	700	200	525	7	177.579	0.6798
✓	184	Completed	5	5 of 5	700	250	500	7	179.139	0.859403
✓	122	Completed	5	5 of 5	700	400	500	7	187.776	0.980905
✓	069	Completed	5	5 of 5	700	475	500	7	191.684	0.989216
✓	067	Completed	5	5 of 5	700	500	500	7	191.684	0.989216
✓	055	Completed	5	5 of 5	700	450	500	7	191.684	0.989216
✓	044	Completed	5	5 of 5	700	450	500	6	220.222	1
✓	030	Completed	5	5 of 5	700	200	500	7	168.805	0.664521
✓	007	Completed	5	5 of 5	700	450	525	7	210.535	0.999095

Figure 10.37: Part of the Points Generated on the Pattern Frontier by SIMIO

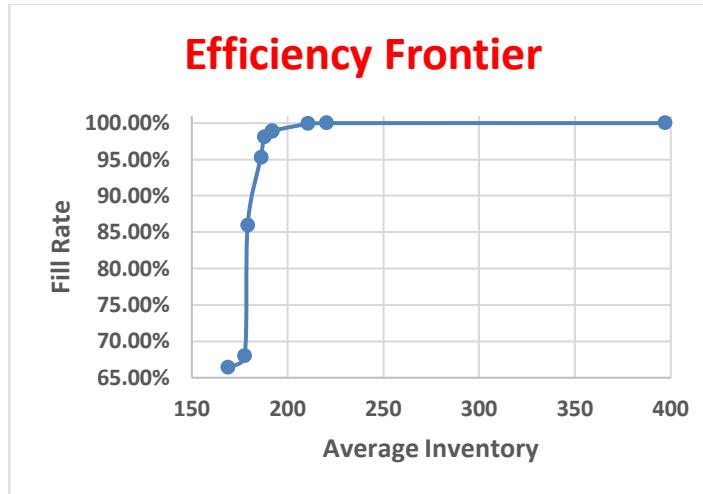


Figure 10.38: Pattern Frontier Graphed in Excel

**Step 9:** Note that more complicated constraints can be composed on the inputs (controls). These constraints are more common. A nonsensical example of a constraint on an input in this model could be ( $\text{ReorderPoint} + 3 * \text{OrderUpToQty} \geq 1000$ ), which is inserted by selecting the *Add Constraint* button and entering the equation as seen in Figure 10.39.

Properties: NonSensicalConstraint (Constraint)	
General	
Name	NonSensicalConstraint
Expression	ReorderPoint + 3 * OrderUpToQty
Lower Bound	1000
Upper Bound	

Figure 10.39: Example of Constraint on Decision Variables

### Part 10.9: Commentary

- The use of the subset selection facilitates finding a good collection of solutions. The KN ranking and selection scheme is quite good for statistically determining the best scenario for many control variables and scenarios.
- OptQuest is an outstanding optimizer, given enough time and replications. However, the SMORES, subset selection, and KN greatly help determine what “solution” you think is best.
- With the Cost Financials added to SIMIO, one could have optimized cost as well.

# Chapter 11

## Simulation Output Analysis

---

You have carefully created a model and paid close attention to the present operational system. Your model produces appropriate performance measures. You may have done some considerable re-modeling as you learned more about the system being modeled and its important aspects. You have been concerned about modeling errors, so you checked your model often and *verified* it is working the way you want. Furthermore, you have addressed the question of model validity by checking the simulated performance measures against actual results. Through all your efforts, you have been thinking about ways to improve the system performance and have an accumulated set of alternative scenarios you want to try. You are now satisfied that you can begin to examine these alternative scenarios and begin the process of developing recommendations.

The simulation model produces outputs, but we know they are sets of random variables. But how do we best understand these results? We can generate averages of observed outputs, but these averages exhibit variation. Can we simply make multiple replications of our model, create confidence intervals, and draw conclusions?

### Part 11.1: What can go wrong?

There are two overriding statistical issues that should concern us. The first is bias in our estimates, which are potentially wrong estimates. The second is the “independence and identicalness” assumptions, which plague us when calculating measures of variability and ultimately comparing systems.

#### Bias

Perhaps the biggest concern is the potential “*bias*” in our computed averages. Bias means that we computed a value that is systematically different than the performance measure of interest. In other words, we may be computing a value that simply isn’t the value of interest. Bias can occur in our method of calculation, but it is more likely in the values being used. For instance, consider the computation of the average waiting time in the queue of the ice cream store. Now, we know that the simulation starts “empty and idle,” so the early waiting times will be considerably smaller than those during the day. In fact, the expected waiting time changes throughout the day. Since the arrival process uses a time-varying arrival, the waiting times grow and shrink with a strong influence from the arrival rate. Other statistics, like resource utilization, will vary as well.

Bias may or may not be a problem. If we need to know the waiting time, say during the busiest time in the ice cream store, computing the waiting time during the entire day will clearly underestimate that waiting time. In such a case, we need to collect our own statistics during the periods of interest. On the other hand, an overall average waiting time computed over the whole day can be useful. While it may underestimate our intuition about the congestion, it is a systematic and consistent computation that can be useful in comparing alternative scenarios. Because the waiting time will be computed in the same manner in all scenarios, the observed improvement or deterioration will be “relatively” correct. In other words, our absolute estimate of waiting times may be an underestimate, but that same degree of underestimation will be present in all scenarios explored.

This idea that relative differences are the important perspective allows us to use simulation models that underestimate many performance measures simply because all the “small” factors that impact the performance measure are being ignored. For instance, our simulation model may only account for 50% of a resource’s duties when we know their workload is closer to 80%. But because all our scenarios ignore those same small factors (e.g., meetings, breaks, interruptions), if we find a scenario that enhances the resource to 60% utilization, then we believe that the base utilization of the resource will be increased by 10% by employing this scenario. So, the recommendation here is to use a relative difference perspective rather than an absolute perspective when possible.

## **Independent and Identical**

In order to measure variability associated with any simulation output, one needs data to be independent and identically distributed (IID). The reason is that the confidence interval computation for an expected value is based on the central limit theorem, which allows us to compute the standard deviation of the mean from the standard deviation of the observation. Unfortunately, variability is present in almost all our simulation output, so we had to “construct” IID observations. Early in our study of simulation, we recognized that the observations within a single replication do not satisfy the IID requirement. So, we rejected the observed waiting times within a simulation replication as unsuitable for estimating the variance of an expected value. Instead, we compute the replication average as an observation, which works for most of the cases of interest.

There are a couple of situations where getting the IID observations may be a cause for concern. One case occurs when a simulation replication takes a very long time. It becomes very costly (in terms of computer time) to run all of the appropriate replications needed. A second case is when the simulation starts from, for example, empty and idle, and there is a lengthy period when the performance measures are poorly estimated due to the startup of the simulation. In both cases, using replications (i.e., one observation per replication) becomes problematic since we have limited opportunity to do many replications.

## **Part 11.2: Types of Simulation Analyses**

There are two general types of simulation analyses: *terminating* and *steady-state*. A terminating simulation typically models a terminating system that has a natural opening and closing. The ice cream store is a good example because it opens empty and idle in the morning and then closes in the afternoon, sometimes after the last customer is served. Terminating simulations are characterized by their opening and closing. Thus, any performance measures computed implicitly depend on how the simulation starts and stops. One replication is generally the time between the opening and closing. A terminating simulation is sometimes called a “transient” simulation because its state is constantly changing throughout the replication. Although bias in the estimates of the averages may occur because of the changing state, the estimation of variance is facilitated by the use of replications.

*Question 1:* What is an example of a simulation that should be examined as a terminating simulation?

---

A steady-state simulation has no “natural” opening or closing. In a steady-state simulation, we want our performance measures to be independent of how we started or stopped the simulation. In other words, we want a kind of long-term average for the expectations. However, while there is considerable appeal for performance measures that are independent of starting and stopping, how to obtain such a simulation is not clear. There are two fundamental problems. The first is how to eliminate the effects of startup or warmup. The influence of the way the simulation starts can extend far into the simulation. The second problem is how long a replication should last since there is no natural stopping state. Neither of these problems can be easily answered, so the tendency is to resort to “rules of thumb” that give some general guidance but for which there is limited theory.

*Question 2:* What is an example of a simulation that should be examined as a steady-state simulation?

---

## **Part 11.3: Output Analysis**

To provide a context for an output analysis, re-consider the work cell problem of Chapter 5, section 6. Recall that this problem is a manufacturing cell consisting of three workstations (i.e., A, B, C) through which four-part types are processed. The parts have their own sequences, and processing times depend on the part type and the station visited. We defined two statistics of interest: a “state statistic” on the number of Part type 3’s in the system during the simulation and a “tally statistic” on the time Part type 3’s are in the system.

Very little analysis of the problem was undertaken. We ran the simulation (interactively) for 40 hours and examined the output. The average number of Part type 3's in the system was 0.40 parts, and the average time for Part type 3's in the system was 0.26 hours. At that time, our focus was on the construction of the model, not necessarily the interpretation of the output, which we will explore now.

Since we have more experience with simulation output, we may be considering a SIMIO experiment and making multiple replications. By making multiple replications, we obtain an estimate of the variability associated with the observed averages and could compute a confidence interval. But before doing that analysis, let's think about our simulation intent. How do we want to use the output from the simulation? Do we want to say something about the "short-term" or the "long-term" outcomes? We probably chose a 40-hour simulation length because it represents a "week's worth" of work. However, we also used the default "empty and idle" state for the start of the week. Is that representative of a real week? If the answer is yes, then we can make multiple replications and create confidence intervals for our output measures.

*Question 3:* In your judgment, how likely are you interested in one week's statistics being empty and idle at the beginning of the week?

---

By doing replications, we should obtain approximate IID observations, but a more fundamental concern is bias. We have two sources of bias. The first is the "initialization" or "warm up." Starting empty and idle certainly affects our statistics. The second is whether the 40 hours is sufficient time to observe the system. To limit our discussion, let's focus primarily on the two user-defined statistics: the number of Part type 3's in the system during the simulation and the time Part type 3's are in the system.

### Initialization Bias

When we describe the initialization bias, we often use the term "warm up," implying that the simulation output goes through a phase change like a car that needs to warm up in cold weather before turning on the heater. We think of a "transient phase" followed by a transition to the "steady-state phase." And usually, the steady-state phase interests us since it represents the "long-term" perspective.

Three "obvious" ways to deal with the initialization bias are: (1) ignore it, trusting that you have enough "good" statistical contributions from the steady-state phase that the "poor" statistics at the beginning (during the transient period) are completely diluted, (2) "load" your system with entities at the beginning of the simulation so that the transition to steady-state behavior occurs early, and (3) after a while (i.e., warm up period), clear (truncate) the statistics in an attempt to clean out the transient statistics so they can no longer influence the statistics collection.

The problem with ignoring the presence of the transient statistics is that their influence on the final statistics is really unknown. The problem with loading the system with entities is that we don't know where the loading should take place and furthermore, we don't know that our loading really hastens the transition to steady-state. Finally, the problem with deleting statistics is that we delete "good" as well as "bad" statistics and simulation observations are important and usually somewhat hard to obtain. So, let's consider a more thorough examination of initialization bias in the context of the work cell problem.

**Step 1:** Open up the simulation from Chapter 5, section 6. For comparison purposes, run the model interactively for 40 hours.

*Question 4:* What is the average number of parts of type three in the system during the 40 hours?

---

*Question 5:* What is the average time parts of type three are in the system during the 40 hours?

---

**Step 2:** Add a SIMIO Experiment, specifying ten replications, each replication being 40 hours. We will also use these results for comparison.

**Question 6:** What is the 95% confidence interval on the average number of parts of type three in the system during the 40 hours?

---

**Question 7:** What is the 95% confidence interval on the average time parts of type three are in the system during the 40 hours?

---

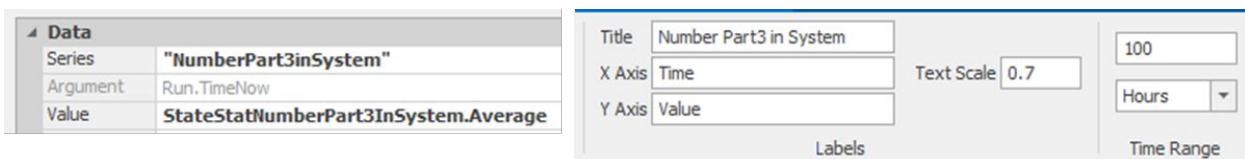
**Question 8:** Do the averages from a single replication fall with the confidence intervals?

---

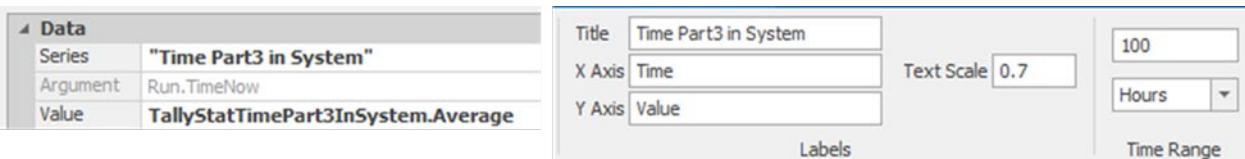
**Question 9:** Why wouldn't you be concerned if they didn't?

---

**Step 3:** Next, let's add some plots of the two user-statistics so we can visualize the change in these over time due to the initialization bias. Keep in mind this simulation starts empty and idle. Add two Animation Plots from the Animation tab: one for the number of parts of type three in the system and the second for the time parts of type three are in the system. Figure 11.1 shows the specifications for the average number of part type three in the system. Figure 11.2 shows the specifications for the average time part type 3 is in the system.



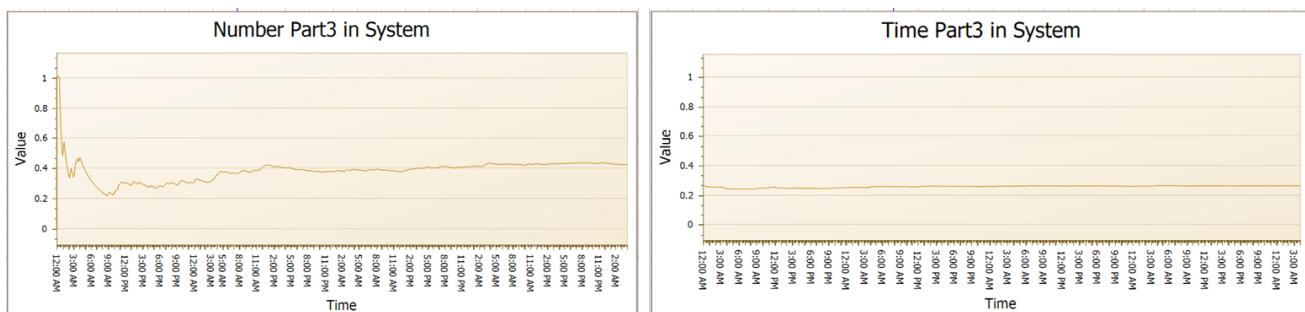
**Figure 11.1: Animation Plot for Average Number of Part Type 3 in System**



**Figure 11.2: Animation Plot for Average Time Part type 3 in System**

**Step 4:** Be sure that the *Time Range* on the plots is 100 hours so we have enough time to see the changes.

**Step 5:** Change the run length of the simulation to 100 hours and, run the simulation interactively (set the speed factor to 1000) and observe the plots.



**Figure 11.3: Average Number and Time in System**

**Step 6:** Notice how the number in system plot changes rather markedly during the first 50 hours, while the time in system plot doesn't change much.

**Question 10:** During the initial 50 hours, how long does the transient period seem to last (recall this is the time during which the behavior is erratic)?

---

**Question 11:** After the transient period, there is a period of transition as the state of the statistics moves from being highly unstable to becoming stable. How long do you think this transition period lasts?

---

**Question 12:** How are the animation plots changing during the last 50 hours?

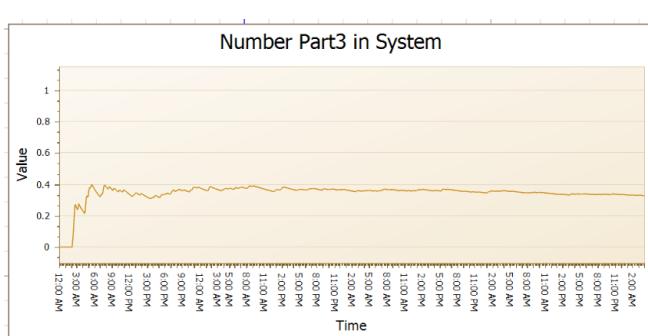
---

When these plots stabilize, we say the statistics are in steady-state. Notice that we talk about the statistics, not the system because it's the statistics that interest us, even though it's the system that is changing.

**Question 13:** Which statistic enters steady-state first? Number in system or time in system?

---

**Step 7:** We have seen only one replication during the 100-hour simulation. Under the Advanced Options within the Run Setup section of the Run tab, click on Randomness and select the Replication Number option. Set the replication number to "2" and re-run the 100-hour simulation. Now, the time in the system continues to be relatively stable, while the number in the system shows a different transient behavior. Figure 11.4 shows the average number of part type three's in the system during the first 100 hours.

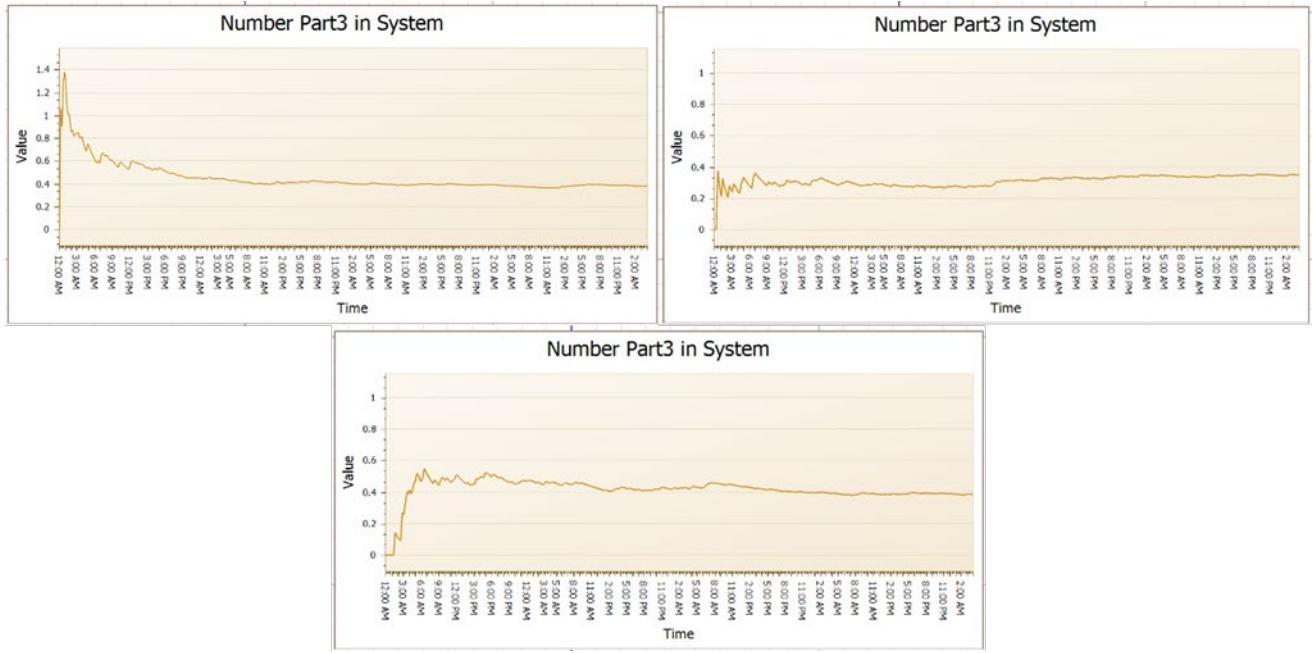


**Figure 11.4: Replication 2 Average Number Part 3 in System**

**Question 15:** How does the transient phase in Replication 2 for the average number of part 3 in the system differ from the one for Replication 1 (which is shown in Figure 11.3)?

---

**Step 9:** Now, continue to change the replication number and interactively simulate, observing the behavior of the average number of part 3's in the system during the first 50 hours. Figure 11.5 shows the results for replications 3, 4, and 5. We focus on number in system, but be sure to look at the time in system also.



**Figure 11.5: Replications 3, 4, and 5 Average Number of Part Type 3's in System**

**Step 10:** With these five replications, we can see a range of behaviors at the beginning of the simulation. In all cases, the statistics appear to converge to a common steady-state value, but there was a consistent transient period during which there was no consistent behavior.

**Step 11:** Now simulate Replication 5 for 40 hours interactively and compare to *Step 1*.

*Question 16:* What is the average number of parts of type three in the system during the 40 hours?

---

*Question 17:* What is the average time parts of type three are in the system during the 40 hours?

---

*Question 18:* In comparing 40 hours with 100 hours simulations, do you think the differences are important?

---

**Step 12:** While the differences are probably not important in this case, you can see that they may be in other cases. In any case, our simulation would benefit by eliminating the initialization bias. Now, the question is, how large should the warmup be? Clearly, we want to eliminate the transient period, which, after observing five different replications, lasts for about 20 hours. The transition period still shows some effect, but we don't want to throw away observations unless we think they are really bad. So, let's make the warmup period 15 hours.

### Replication Length

Now, as a rule of thumb, we do not want the warmup period to exceed 10-20% of the length of a replication. In fact, if time permits, 5% would be a target.

**Step 13:** So, let's run the simulation experiment with a run length of 150 hours with a warmup of 15 hours with a total of 10 replications.

*Question 19:* What is the 95% confidence interval on the average number of parts of type three in the system?

---

*Question 20:* What is the 95% confidence interval on the average time parts of type three are in the system?

---

*Question 21:* How do these confidence intervals compare to the ones from *Step 2* previously?

---

In summary, removing the initialization bias by using the warmup period is important. However, you should look at several replications of the warmup period in order to understand the range of behavior. Also, you need to examine all the statistics of interest so your warmup covers the largest transient behavior. With the warmup period established, be sure that the warmup period is less than 20% of the replication length. It is generally a good rule to multiply the warmup period by 10 to obtain the replication length. Use a multiplier of 20 if your model runs quickly enough.

#### Part 11.4: Automatic Batching of Output

To obtain our previous confidence interval, we simulated (150 hours \* 10 replications) for a total of 1500 hours. Of that total, we used 450 hours for the warmup. Although this small problem runs through that simulation time rather quickly, a larger problem may take hours of computer time to provide such a confidence interval. So, we may be interested in obtaining a confidence interval in less time. We “wasted” many observations during warmup and created fairly lengthy simulation replications.

An alternative to replications is employing a method known as “**batch means**.” Using batch means, we attempt to create IID observations within one long replication. Since only one run of the simulation is used, we only lose the warmup observations once, and we are able to invest our simulation effort in one long run rather than several. Hopefully, all this means that we save simulation time. The batch means method creates “batches” of observations for which the average for the batches satisfies our IID requirement. For example, the time-persistent statistics (state statistics), like the number in the system, may have a mean computed every 25 hours if the means from every 25 hours appear to be IID. For observations-based statistics (tally statistics), we might compute an average waiting time for every 50 customers if the means from every 50 customers appear to be IID. The 25 hours for state statistics and the 50 customers for tally statistics are called the “**batch size**.” There will be two batch sizes: one for state statistics and one for tally statistics. The number of batches created during the simulation corresponds to the number of observations when computing a confidence interval.

SIMIO will automatically employ a version of batch means to compute a confidence interval when: (1) user statistics are defined, and (2) an SIMIO experiment with one replication is employed. The only SIMIO-defined statistics that will be automatically batched are the sink node statistics.

**Step 1:** Use the simulation model from the previous section. Change the number of required replications to one. Leave the warmup at 15 hours and leave the replication length at 150 hours.

**Step 2:** When you execute the simulation, you will see a new row called “HalfWidth” in the two user-defined statistics for the average number of part type 3 in the system and the average time part type 3 is in the system. You will also see that new row in the **SnkPartsLeave** statistics on flowtime. The output for the two user-defined statistics is shown in Figure 11.6.

Model	Model	StateStatNumberPart...	UserSpecified	StateValue	Average	0.4347	0.4347	0.4347	NaN
			HalfWidth Row		FinalValue	0.0000	0.0000	0.0000	NaN
			HalfWidth Row		HalfWidth	0.0594	0.0594	0.0594	NaN
			HalfWidth Row		Maximum	6.0000	6.0000	6.0000	NaN
			HalfWidth Row		Average (Hou...	0.2667	0.2667	0.2667	NaN
		TallyStatTimePart3In...	UserSpecified	TallyValue	HalfWidth (Ho...	Insufficient	Insufficie	Insufficien	NaN
			HalfWidth Row		Maximum (Ho...	0.5472	0.5472	0.5472	NaN
			HalfWidth Row		Minimum (hou...	0.1674	0.1674	0.1674	NaN
			HalfWidth Row		Observations	220.0000	220.0000	220.0000	NaN
			HalfWidth Row						

**Figure 11.6: Output for User-Defined Statistics**

**Step 3:** In the case of the time in system we see an “Insufficient,” meaning we have run the simulation long enough for SIMIO to obtain enough batches to compute a confidence interval. It is also possible to get a “Correlated” message meaning that the obtained batches do not satisfy the IID requirement. Figure 11.7 is an example of obtaining this message.

Model	Model	StateStatNumberPart...	UserSpecified	StateValue	Average	0.2318	0.2318	0.2318	NaN
			HalfWidth Row		FinalValue	1.0000	1.0000	1.0000	NaN
			HalfWidth Row		HalfWidth	Correlated	Correlate	Correlate	NaN
			HalfWidth Row		Maximum	2.0000	2.0000	2.0000	NaN
			HalfWidth Row						

**Figure 11.7: Example of Correlated Batch Means**

*Question 22:* Why do the columns for “Average,” “Minimum,” and “Maximum” all have the same value?

---

*Question 23:* Why does the “HalfWidth” show “NaN” (Not a Number)? Remember, we are using only one replication.

---

**Step 4:** Now, change the replication length to 300 hours, which is twice the length of the earlier replications.

*Question 24:* Do you obtain any “Insufficient” or “Correlated” messages on the statistics?

---

*Question 25:* What is the 95% confidence interval on the average number of parts of type 3 in the system?

---

*Question 26:* What is the 95% confidence interval on the average time parts of type 3 are in the system?

---

*Question 27:* How do these confidence intervals compare to the ones from **Section 3** previously?

---

The automatic batching of statistics by SIMIO was able to produce a “legitimate” confidence interval with only 300 hours of simulation, as opposed to the 1500 hours previously. When using automatic batching, we still needed the warmup analysis.

### Part 11.5: Algorithms used in SIMIO Batch Means Method

SIMIO automatically attempts to compute a confidence interval via a batch means method for only the user-defined tally and state statistics (the only exception is that the batch means method is also applied to sink statistics) for a SIMIO Experiment having only one replication. It displays the results in the output in the **row** called “HalfWidth,” which is one-half the confidence interval. It can fail to make the computation if it believes there is insufficient data or if the batched data remains correlated. SIMIO does not include data obtained during the *Warmup Period* specified in the Experiment.

SIMIO's batching method is done "on the fly," meaning the batches are formed during the simulation. Thus batches can be enlarged but not re-batched.

### Minimum Sufficient Data

- For a Tally statistic, SIMIO requires a minimum of 320 observations
- For a State statistic, there must be at least 320 changes in the variable during at least five units of simulated time.
- For statistics that do not satisfy this minimum requirement, the half width is labeled "Insufficient."

### Forming Batches

- Form 20 batch means
  - A Tally batch is the mean of 16 consecutive observations
  - A State batch will be the time average over 0.25 time units
- Continue forming batches until there are 40 batches
  - At 40 batches, the batch size is doubled (each batch is "twice as big"), and the number of batches is reduced to 20
- Continue this procedure as long as the simulation runs. Notice that the number of batches will be between 20 and 39

### Test for Correlation

- SIMIO uses VonNeuman's Test<sup>149</sup> on the final set of batches to determine if the assumption of independence can be justified.
  - If the test fails, the half-width is labeled "Correlated."
  - If the test passes, the half-width is computed and displayed.

### Cautions

The SIMIO batch means the method is only for steady-state analysis – it should be ignored entirely for terminating systems. There is no automatic determination of the warmup period. All rules for collecting batches are somewhat arbitrary, so some care needs to be exercised when using them.

## Part 11.6: Input Analysis

It may seem strange to include "input analysis" in a chapter devoted to "output analysis," but it must be remembered that the analysis of the input is based on the behavior of the output. SIMIO provides two input data analyses: (1) **Response Sensitivity Analysis** and (2) **Sample Size Error Analysis**. Both analyses require that at least one of your inputs be an *Input Parameter* and that you have defined at least one *Experiment Response*. Although we briefly introduced response sensitivity analysis previously, we will describe it as well as the sample size error analysis within the same context, namely the work cell problem used throughout this chapter.

Recall that "input parameters" are defined from the "Data" window tab in three possible ways: (1) through a **DISTRIBUTION** you hypothesized, (2) with a **TABLE** of observations that you have collected, and (3) using a **SIMIO EXPRESSION**. Any defined input parameter can be used anywhere in your model, and various objects can reference the same input parameter. Using distributions has the advantage that SIMIO will provide you with a histogram based on 10,000 samples to show you what that distribution looks like.

**Step 1:** Save the current model and delete the animation plots, as they will not be needed.

---

<sup>149</sup> This is a statistical test of independence.

**Step 2:** Rename the model state variable to **GStaNumberInSystem**. Modify the two user-defined statistics so we have some possible responses for experimentation. Call the state statistic **StateStatNumberInSystem** and the tally statistic **TallyStatTimeInSystem**. Be sure that the state statistic references the **GStaNumberInSystem** state variable and that the tally statistic has Time as the *Unit Type*.

**Step 3:** Fix the *State Assignment* at the **SrcParts** and the **SnkPartsLeave** so that **GStaNumberInSystem** is always incremented (add one) at the source and decremented (subtract one) at the sink. Modify the *Tally Statistics* collection at the **Input@SnkPartsLeave** to correspond to Figure 11.8.

Basic Logic	
Tally If	Entity Entering
Tally Statistic Name	<b>TallyStatTimePartInSystem</b>
Value Type	Expression
Value Expression	<b>ModelEntity.TimeInSystem</b>

Figure 11.8: Tally All Entities Time in System

**Step 5:** Next, we need to create the input parameters, which will correspond to, in this case, all the distributions used in the model. These are created by specifying a “*Distribution*” with the *Input Parameters* section from the “*Data*” Tab. Make sure to change the *Unit Type* property to “Time” and specify minutes. Figure 11.9 shows the inputs, and these are the same as the **SequencePart** table.

Name	Object Type	Summary
<b>Input Parameters</b>		
InpDistPartsArrive	Distribution Input Parameter	Exponential(10 Minutes)
InpDistPart1StationA	Distribution Input Parameter	Pert(2 Minutes,5 Minutes,8 Minutes)
InpDistPart1StationC	Distribution Input Parameter	Pert(2 Minutes,6 Minutes,11 Minutes)
InpDistPart2StationA	Distribution Input Parameter	Pert(1 Minutes,3 Minutes,4 Minutes)
InpDistPart2StationB	Distribution Input Parameter	Uniform(5 Minutes,11 Minutes)
InpDistPart2StationC	Distribution Input Parameter	Uniform(2 Minutes,11 Minutes)
InpDistPart3StationA	Distribution Input Parameter	Triangular(2 Minutes,5 Minutes,8 Minutes)
InpDistPart3StationB	Distribution Input Parameter	Triangular(5 Minutes,9 Minutes,11 Minutes)
InpDistPart4StationB	Distribution Input Parameter	Pert(5 Minutes,9 Minutes,11 Minutes)
InpDistPart4StationC	Distribution Input Parameter	Triangular(2 Minutes,6 Minutes,11 Minutes)

Figure 11.9: Definition of Input Parameters

**Step 6:** Select the **SrcParts** and use the **InpDistPartsArrive** as the *Interarrival Time*.

**Step 7:** Also, substitute the appropriate input distribution into the **SequencePart Data Table** for each of the **ProcessingTimes** values, as seen in the partial view of the table in Figure 11.10.

Table Parts		Sequence Part	
	Sequence	ProcessingTimes (Minutes)	ID
1	Input@SrvStationA	InpDistPart1StationA	0
2	Input@SrvStationC	InpDistPart1StationC	0
3	Input@SnkPartsLeave	0	0
4	Input@SrvStationA	InpDistPart2StationA	1
5	Input@SrvStationB	InpDistPart2StationB	1
6	Input@SrvStationC	InpDistPart2StationC	1
7	Input@SnkPartsLeave	0	1

Figure 11.10: Partial View of the Sequence Part Table using the New Input Distributions

**Step 8:** Next, select the **Experiment** and add two experiment responses, as shown in Figure 11.11.

Properties: NumberInSystem (Response)		Properties: TimeInSystem (Response)	
General		General	
Name	NumberInSystem	Name	TimeInSystem
Display Name	NumberInSystem	Display Name	TimeInSystem
Description		Description	
Expression	StateStatNumberPartInSystem.average	Expression	TallyStatTimePartInSystem.Average
Unit Type	Unspecified	Unit Type	Time
Objective	None	Display Units	Minutes
		Objective	None

**Figure 11.11: Experiment Responses for Number and Time in System**

**Step 9:** Run the experiment with a *Warmup Period* of 15 hours and a run length of 150 hours with 20 replications, which will be used as a base reference for the results.

**Question 28:** What did you get for the 95% confidence interval on the average number in the system?

---

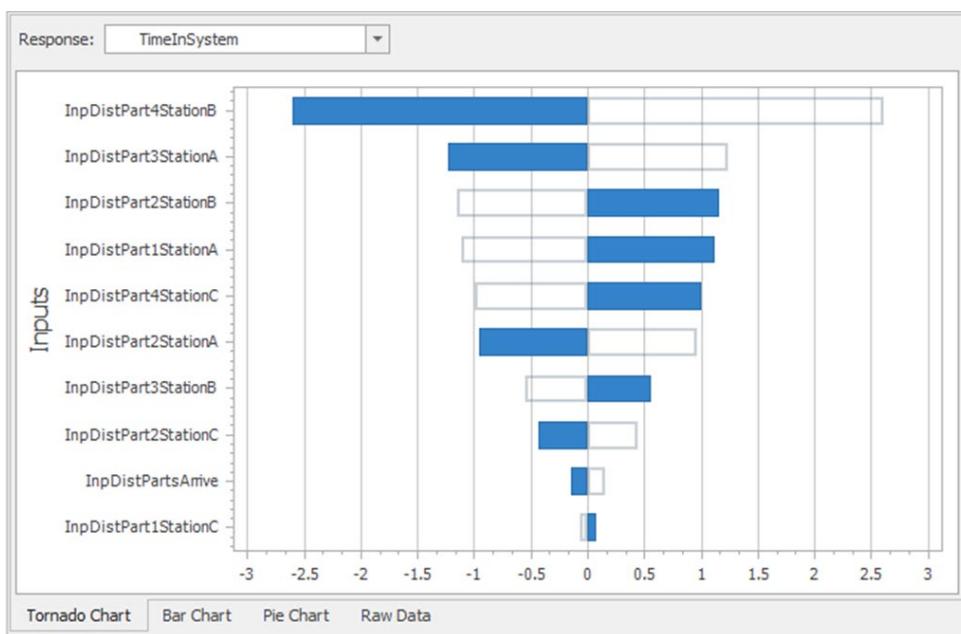
**Question 29:** What did you get for the 95% confidence interval for the average time in the system?

---

## Response Sensitivity Analysis

The response sensitivity analysis measures how each response is affected by changes in the input parameters. It will be computed for each scenario in the experiment. A linear regression relates each input parameter to each response, so the number of replications must be greater than the number of input parameters.

**Step 10:** Our prior model had ten input parameters, and we used 20 replications. Select the *Input Analysis* tab and then the *Response Sensitivity* icon and run the model (it may be necessary to Reset it first). The results of the sensitivity analyses are shown in a Tornado Chart, a Bar Chart, and a Pie Chart. The raw data for each of the regressions are also available.



**Figure 11.12: Tornado Chart for Time in System**

**Step 11:** Refer to the Tornado Chart in Figure 11.12 and pass your cursor over the bars. If an input parameter has a negative coefficient, then when it is increased, it reduces the response. If the input

parameter has a positive coefficient, then when it is increased, it increases the response. The magnitude of the coefficient determines its importance relative to the other input parameters.

*Question 30:* What input parameter will reduce the time in system the most when its distribution is increased?

---

*Question 31:* What input parameter will increase the time in system the most when its distribution is increased?

---

*Question 32:* What input parameter reduces the number in system the most when its distribution is increased?

---

*Question 33:* What input parameter increases the number in system the most when its distribution is increased?

---

*Question 34:* What surprises you about these results?

---

*Question 35:* What is the primary value of the pie chart relative to the sensitivity analysis?

---

—

### Sample Size Error Analysis

The sample size error analysis attempts to assess the uncertainty in the responses relative to the uncertainty in the experiment and the uncertainty in the input parameter data. To perform this analysis, it is assumed that the input parameters were estimated from collected data samples. In other words, you have a record of observations you used for a given input parameter estimation (or distribution fitting).

**Step 12:** Pretend you have collected data for the input parameter estimation. From the Mode and the *Input Parameters* on the “Data” tab, select a defined input parameter. Change the *Number of Data Samples* property according to Table 11.1 and be sure that the *Include Sample Size Error Analysis* property is **True**.

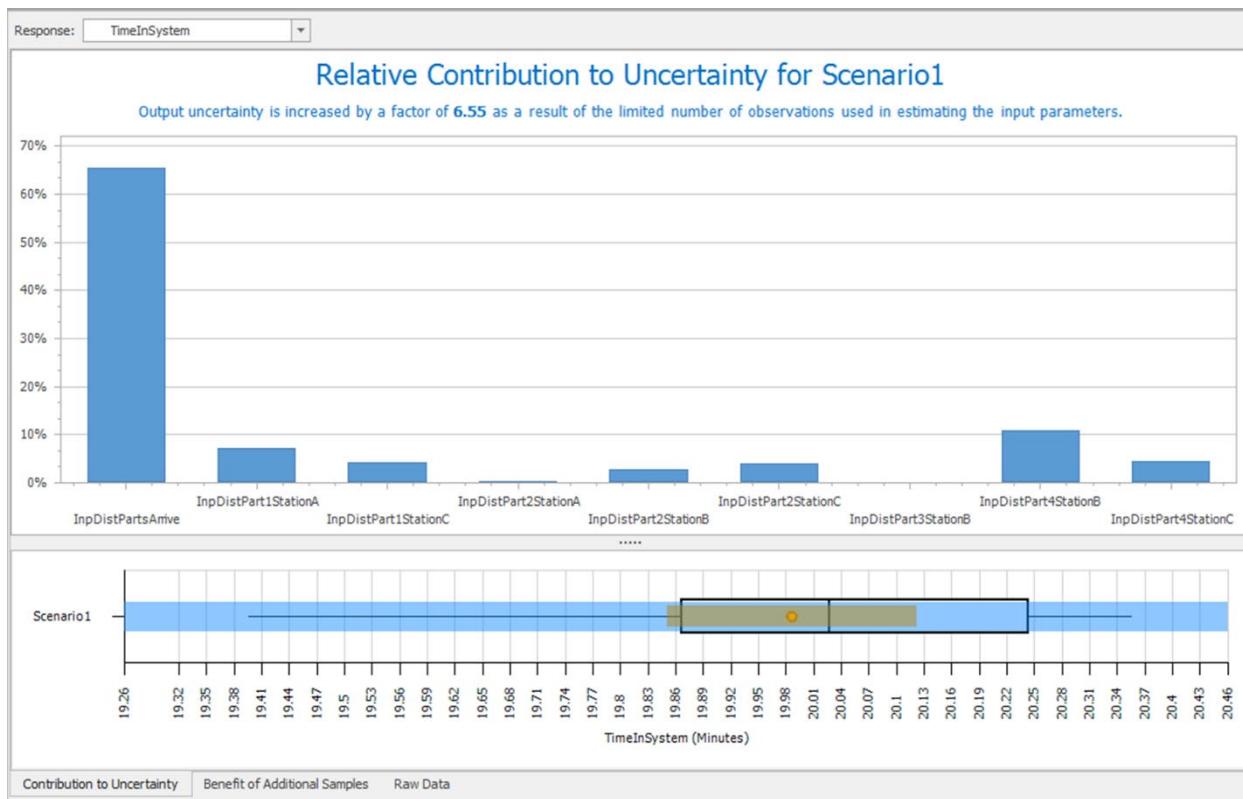
**Table 11.1: Data Samples for Each Input Parameter**

Input Parameter	Data Samples
DistPartsArrive	125
DistPart1StationA	85
DistPart1StationC	85
DistPart2StationA	100
DistPart2StationB	100
DistPart2StationC	100
DistPart3StationA	60
DistPart3StationB	60
DistPart4StationB	45
DistPart4StationC	45

**Step 13:** Go to the *Input Analysis* tab within the EXPERIMENT. Select the *Sample Size Error* icon and run the analysis (after doing a Reset). This may take a little bit of time to complete.

**Step 14:** Next, the experiment will be run from the design view to obtain a SMORE plot.

**Step 15:** The two important tabs are: (1) Contribution to Uncertainty and (2) Benefit of Additional Samples. Look first at the contribution to uncertainty, shown in Figure 11.13.



**Figure 11.13: Uncertainty in Time In System**

**Question 36:** Which factor has the biggest impact on time in the system?

---

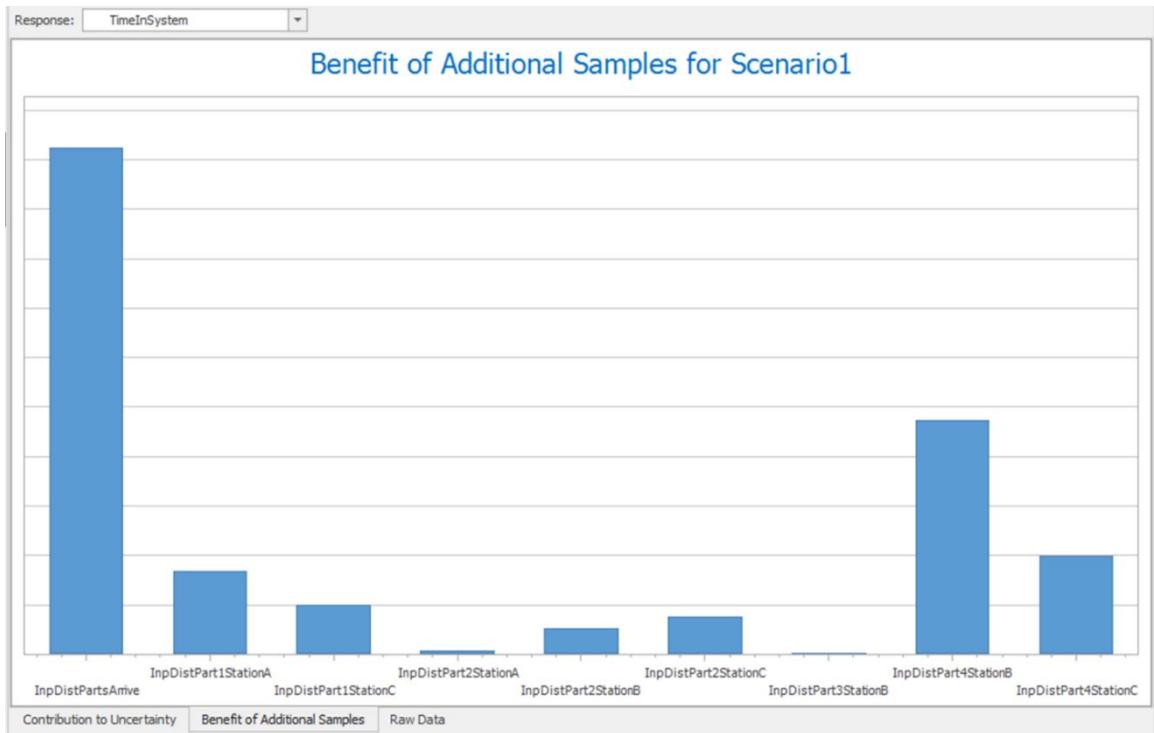
**Question 37:** The SMORE plot illustrates the confidence interval for the mean (in tan), with the expanded half width due to input uncertainty in blue (and stated in the text). How much does the uncertainty in the input parameters “expand” the output uncertainty?

---

**Question 38:** Which uncertainty, input or output, dominates the overall uncertainty in this scenario?

---

**Step 16:** Next, look at the tab: Benefit of Additional Samples as shown in Figure 11.14.



**Figure 11.14: Benefit of Additional Samples**

*Question 39:* Which input parameter would benefit the most from collecting additional observations for time in the system?

---

**Step 17:** Go back to the Input Parameters and change the *Number of Data Samples* property for **InpDistPartsArrive** to 250. Next, repeat the sample size error analysis.

*Question 40:* How does the Contribution to Uncertainty change?

---

*Question 41:* What happens to the Benefit of Additional Samples?

---

*Question 42:* Would you now put more time into observing **InpDistPartsArrive** or would you allocate observations to other input parameters

---

### Part 11.7: Commentary

- Sometimes, the modeling effort consumes so much of a simulation project that the analysis of the model is almost overlooked. In particular, it is easy to forget that simulation output can be difficult to interpret and requires time and care. The issue of bias is often ignored, which can be a critical mistake.
- Replications permit the computation of confidence intervals, which are our best way to measure the mean and variance of the expected value.
- Our experience is that most simulations can be run quickly, which means that eliminating the warmup and replications becomes feasible even for steady-state simulations. However, some steady-state simulations take excessive computer time, so the idea of using batch means becomes attractive.
- The availability input analysis features in SIMIO should encourage you to consider the importance of the simulation inputs relative to your performance measures. You can judge the value of the various inputs and consider the amount of data collection devoted to their specification.

# Chapter 12

## Materials Handling

---

You may have already noticed a Vehicle object and a Conveyor path in the Standard Library. Perhaps the term “vehicle” already conjures up the idea of moving materials or people from place to place, and your intuition would be correct. The VEHICLE object can pick up and delivery entities which can be used to act as a forklift truck in a manufacturing operation, a human stock handler in a warehouse, a taxi cab for picking up and delivering people, as well as a bus, a train, or other type of people mover to move entities either individually or in a group. The VEHICLE is an individual object whose routing is based on either “demand” or a “fixed path.” A conveyor in SIMIO, on the other hand, provides a continuous platform for conveying items or people. It can represent various industrial devices like gravity conveyors, belt conveyors, powered overhead conveyors, and even a power and free conveyor. Often, materials handling in the industry employs a variety of materials handling devices. A distinction among conveyors used by SIMIO is whether the conveyor “accumulates” or doesn’t (“non-accumulating). An accumulating conveyor allows entities to queue up at its end. A non-accumulating conveyor will stop when an entity gets to the end and won’t start up until that entity is removed.

A key to modeling materials handling problems in SIMIO is using the TransferNode and, to a lesser extent, the external OutputBuffer attached to many objects like the SOURCE and the SERVER. The TRANSFERNODE (unlike the BASICNODE) provides the “*Transport Logic*,” which means you can request a vehicle to move the entity to its next designation. If that transport device is not available, then the entity must wait. The TRANSFERNODE can be used alone, and entities can wait for pickup at that node, but they will not queue up in the usual fashion. For other objects, the OUTPUTBUFFER provides a place for the entities to wait for transportation.

### Part 12.1: Vehicles: Cart Transfer in Manufacturing Cell

We will reconsider the original problem of Chapter 5, whose layout and information are shown in Chapter 5, Figure 5.1. Recall that there are five different types of parts, each with its own routing through the manufacturing cell. However, we now learn that instead of leaving the system, parts actually have to be put on a cart and wheeled to the warehouse building 300 yards away (see Figure 5.9). The cart can carry a maximum of three parts and travels four feet per second. After the cart has finished dropping off parts, it will return to the pickup station for the next set of parts or wait until a part is ready to be taken. The cart does not have to be filled to be taken to the warehouse (i.e., it will carry one part if available).

**Step 1:** Open up the last model from Chapter 5 and save it as **chap12.1**.

**Step 2:** Delete the CONNECTOR between the SINK object and the prior BasicNode to facilitate a cart that takes the parts to the warehouse. Next, insert a TRANSFERNODE named **TnodeCart** between the **SnkPartsLeave**, as seen in Figure 12.1.

**Step 3:** Select the **Input@SnkPartsLeave** and remove the *Tally Statistics* as VEHICLES will be entering instead, and we do not want them contributing to the statistic. We could insert a Tally step in the Entered add-on process trigger to calculate the same statistics.

**Step 4:** Connect the BASICNODE to the TRANSFERNODE with a PATH with a logical length of 10 yards. Finally, connect the **TNodeCart** TRANSFERNODE to the SINK with a PATH, ensuring its *Type* is set to “*Bidirectional*” so our cart can move in both directions. Set the path’s logical length to 300 yards.

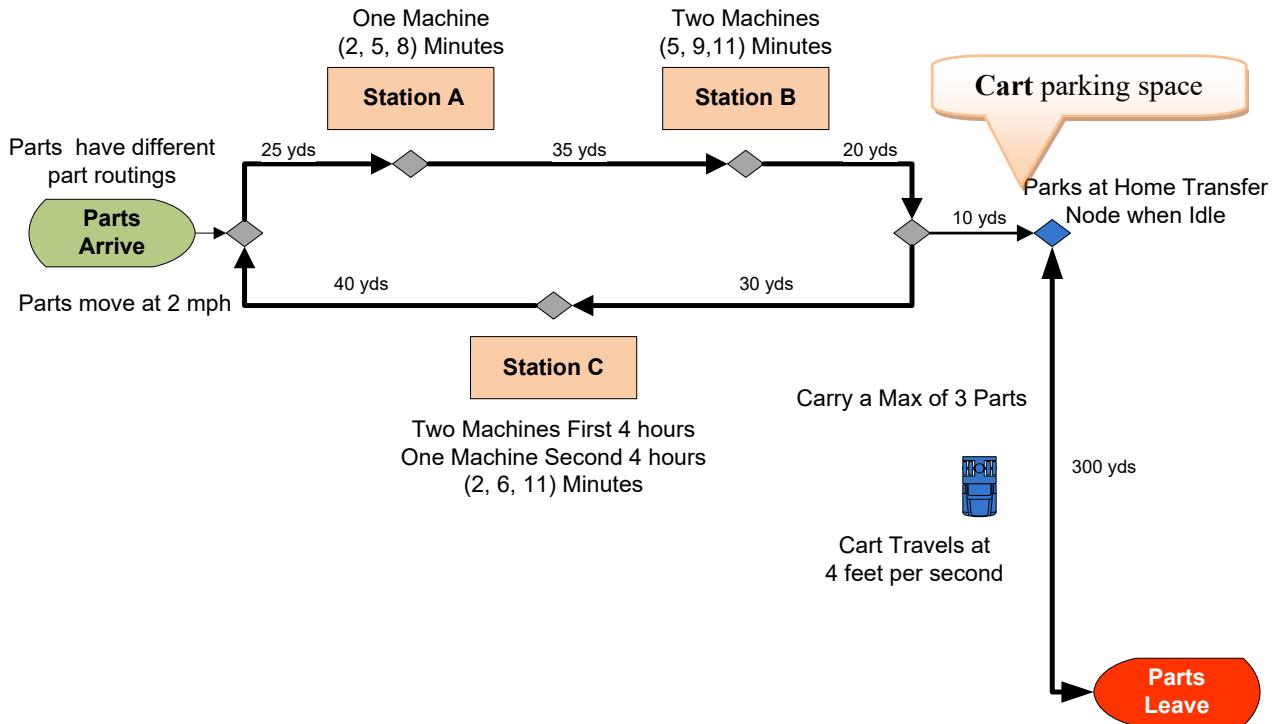


Figure 12.1: Adding a Vehicle to Carry Part to the Exit

**Step 5:** Drag and drop a VEHICLE object from the [Standard Library] to the modeling canvas in the facility window named **VehCart**, similar to Figure 12.2. Make sure to increase the size of the RIDEQUEUE queue, which animates the parts being transported so that it can hold three parts.

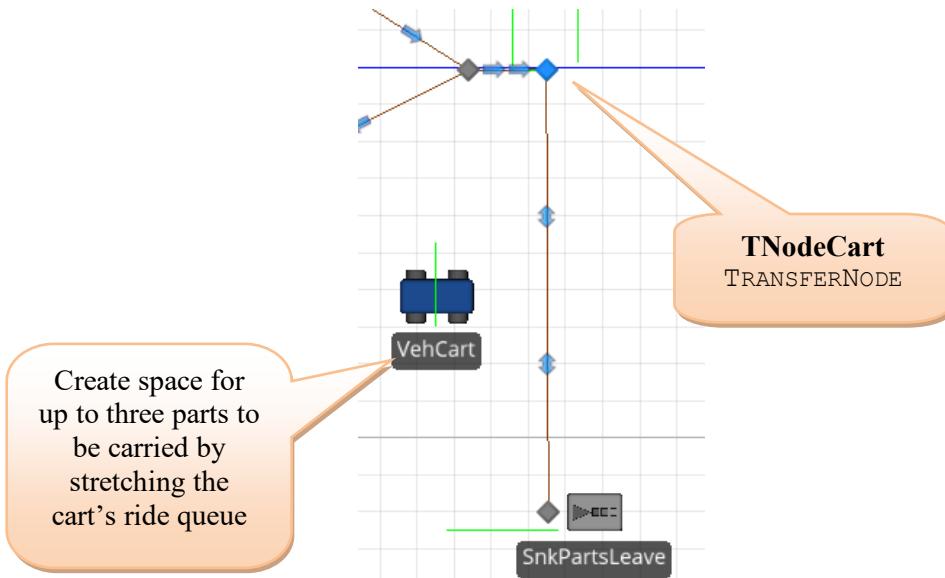
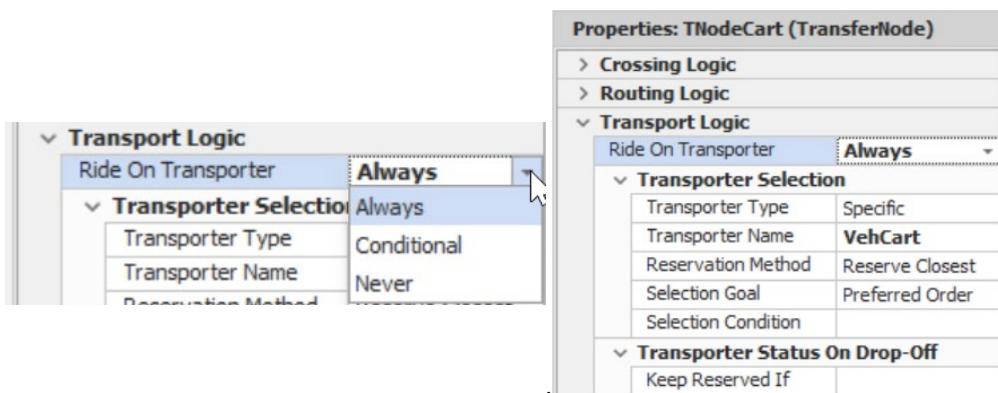


Figure 12.2: Adding a Vehicle

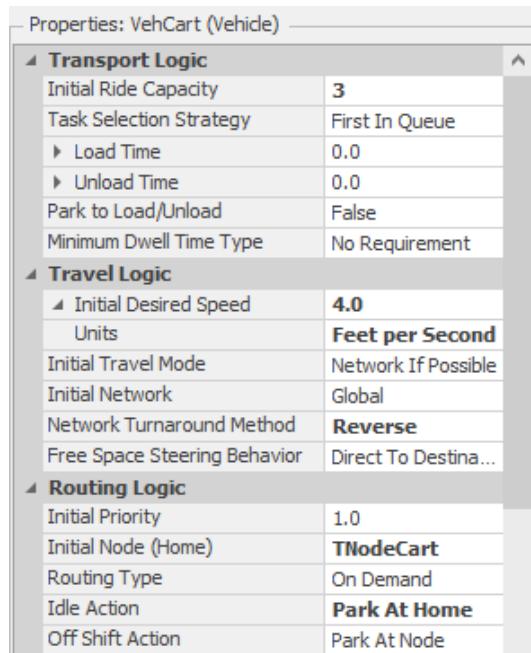
**Step 6:** Change the **TNodeCart** TRANSFERNODE's *Ride On Transporter* property to "Always," meaning entities must always need a vehicle to move. The other option is "Conditional," which allows you to specify a condition that must be true to require a transporter or the entity can move independently. Next, specify the "VehCart" as the *Vehicle Name*, which was just added (see Figure 12.3). Every time an entity

passes through this node, it generates a ride request to be sent to the vehicle. The entity cannot continue until the vehicle is there to pick it up.



**Figure 12.3:** Setting up the Node to Have Entities Utilize a Vehicle

**Step 7:** Use Figure 12.4 as a guide to set the **VehCart**'s *Initial Desired Speed* to four feet per second and *Ride Capacity* to three. Also, set the *Initial Node(Home)* to the **TNodeCart** node, which was just added, as this will be our vehicle's home node. Leave the *Routing Type* specified as “*On Demand*,” meaning it will respond to requests for transportation from the stations as needed. Now, change the **VehCart**'s *Idle Action* property to “*Park At Home*” so that the vehicle will always go to the home node if there are no entities with ride requests.<sup>150</sup> Otherwise, the vehicle will remain at the **SINK** until a ride request is generated.



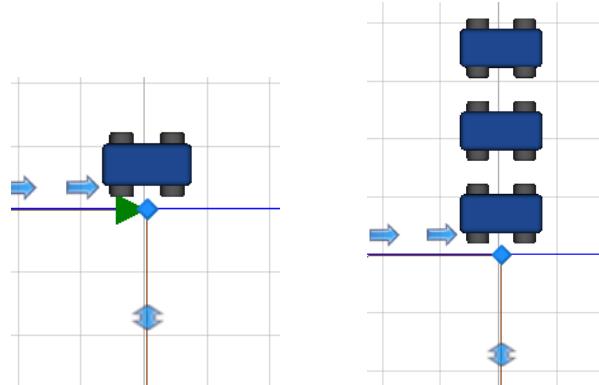
**Figure 12.4:** Vehicle Properties

**Table 12.1:** Process Type Option Explanation

<sup>150</sup> The *Network Turnaround Method* can be changed for example to “Reverse” so the cart would go forward to the exit and then look as if it backs up to the **TNodeCart** node. By default it will turn around and move forward back to the home **TNodeCart** node.

Process Type	Description
Initial Node (Home)	The node where the vehicle will start at the beginning of the simulation.
Routing Type	There are two different routing behaviors. The default “On Demand” option allows the transporter to act like a taxi and only move to pick up when an entity requests a ride. This mode will also enable the transporter to be used as a moveable resource that is seized and released. The “Fixed Route” type requires a sequence table for the transporter to transverse, like a bus route, to pick up entities on the route.
Idle Action	This property specifies what the transporter will do when it becomes idle (i.e., no transport or seize requests) when using the “ <i>On Demand</i> ” routing type. <i>Go to Home</i> and <i>Park at Home</i> . Send the vehicle to the home node, and one will force the vehicle to park in the station rather than visit the node. <i>Remain In Place</i> and <i>Park at Node</i> will do the same thing as the previous two but at the current node. The <i>Roam</i> action will have the transporter roam randomly through the simulation networking while waiting for an on-demand request.
Off Shift Action	It specifies what will happen when the transporter goes off shift and has the same options as the Idle Action except for the no “ <i>Roam</i> ” method.

**Step 8:** Run the simulation now to see how the cart works in the animation. As you can see, the **VehCart** is initially parked at the **TNodeCart** node Figure 12.5. By default, the parking animation queue is above the node<sup>151</sup> and oriented from left to right. If there were more than one vehicle, they would be stacked on top of one another again, facing left to right in the facility, as seen in the right picture in Figure 12.5, where three carts are in the system.<sup>152</sup>



**Figure 12.5: Vehicle is parked at the Home Node using Default Parking Station**

*Question 1:* Does the Cart pick up more than one part?

---

*Question 2:* What do you notice about how a single part or two parts ride on the vehicle?

---

<sup>151</sup> Note the familiar green line that represents the animated queue line is not visible in this case.

<sup>152</sup> Unlike entities, the default number of vehicles is one and can be changed in the Population properties. Also, a Source can create vehicles.

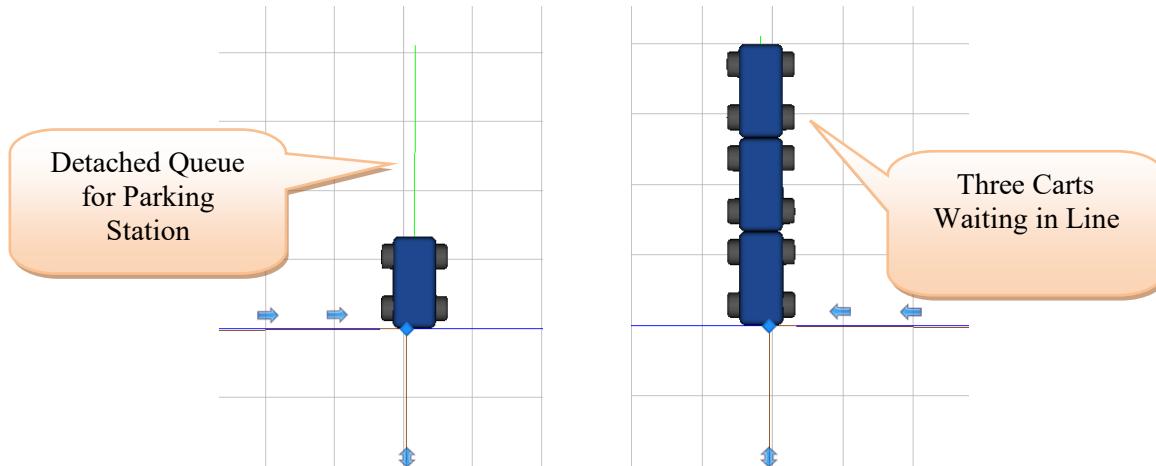
**Step 9:** To fix the issue of a single part riding out to one side, select the RIDE STATION queue on the VEHICLE and change the *Alignment* from “None” to “Point” under the *Appearance* tab. Add an additional vertex to the queue to accommodate three parts and move the first orientation point from the bottom to the middle, as seen in Figure 12.6.



**Figure 12.6: Changing the Orientation of the Riding Queue**

**Step 10:** Run the simulation now to see how the cart works in the animation.

**Step 11:** By default, a parking location for a node is automatically added above the node, as seen in Figure 12.7. However, you should generally not utilize the default one since the location and/or shape of the parting station cannot be adjusted, and the orientation of the vehicles cannot be adjusted. Select the **TNodeCart** node by clicking it and then toggle the *Parking Queue* option in the *Appearance* tab to make it clearer. Next, a parking location is needed for the vehicle, or the vehicle will vanish when parked.<sup>153</sup> Click on the **TNodeCart** node and select “Draw Queue” from the *Attached Animation Tab*. Click on the *ParkingStation*.*Contents*. A cross cursor will now appear in the facility window. Left-click where you want the front of the queue to be and then right-click where you want it to end (before you right-click to end the queue, you could also continue left-clicking to add more vertices). Place it above the node so the vehicle will face downward, as seen in Figure 12.7. Click on the queue and the properties window to ensure the *Queue State is* *ParkingStation*.*Contents*. This queue will animate the parking station of the **TNodeCart** TRANSFERNODE for vehicles parked at the node. It can be oriented and placed at any location. Figure 12.7 shows the same scenario as Figure 12.5, but this time, we can orient the vehicles to point in the same direction as the path.



**Figure 12.7: Vehicle is parked at the Home Node using the New Station**

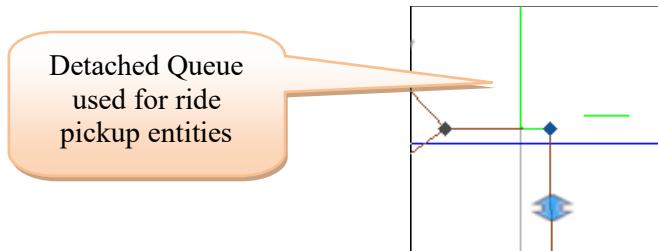
**Step 12:** Run the simulation now to see how the cart works in the animation.

---

<sup>153</sup> The parking queue is again just for the animation since the vehicle is physically at the node even though it cannot be seen.

The model does not show how many parts are waiting for the cart at the transfer node, so it may be helpful to see how many are there at any given time in the simulation. To accomplish this, we can add an animated “Detached Queue” for the **TNodeCart**.

**Step 13:** Click on the **TNodeCart** node and select “Draw Queue” from the “Appearance” Tab. Click on the “RidePickupQueue.”<sup>154</sup> A cursor will now appear in the facility window. Left-click where you want the front of the queue to be and then right-click where you want it to end (before you right-click to end the queue, you could also continue left-clicking to add more vertices). As seen in Figure 12.8, arrange this detached queue in an “L” shape. Otherwise, an entity trying to enter the transfer node will be visible, and overlaying the “L” will show all the entities waiting for transportation.<sup>155</sup>



**Figure 12.8: Ride Pickup Queue**

**Step 14:** Run the simulation to see how the DETACHED QUEUE works.

*Question 3:* What is the utilization of the **VehCart** during a 40-hour replication?

---

## Part 12.2: Cart Transfer among Stations

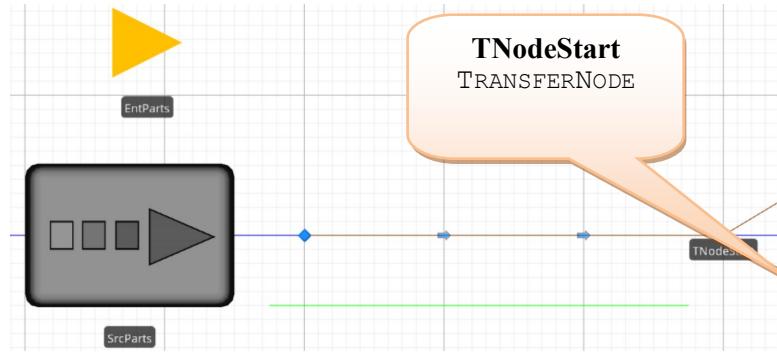
Suppose that the transportation of parts among the stations, including entry and exit, is handled by two similar vehicles. These vehicles are different from the **VehCart**, which carries the parts to the warehouse **SINK**. These carts will wait at the beginning of the circular network until they are needed.

**Step 1:** Convert the one **BASICNODE** that connects the four **SOURCES** into a **TRANSFERNODE** by right-clicking and selecting the *Convert to Type* menu item. This will be necessary for parts entering the network from the sources to request transportation on the inside carts. Next, name this new **TRANSFERNODE TNodeStart**.

---

<sup>154</sup> If added just generic animated queue now, change the Queue State property to **TNodeCart.RidePickupQueue** where **TNodeCart** is the name of the transfer node where the parts are being picked up. Your transfer node’s name may be different.

<sup>155</sup> These queues are for animation purposes only. The parts physically reside at the entrance of the node and one will always be visible as the others will stack up underneath. In later chapters, the notion of a **STATION** to hold entities will be discussed.



**Figure 12.9: Changing the Start Node to a TRANSERNODE**

**Step 2:** Now, add the inside carts by inserting another vehicle from the [Standard Library] into the model and naming it **VehInsideCart**. Shrink the vehicle leaving the ride station queue in the same position.

**Step 3:** Make the **VehInsideCart** have an initial *population* of two<sup>156</sup>, allowing you to see the simultaneous behavior of multiple vehicles on the same path. The new vehicle will have the characteristics shown in Figure 12.10 (i.e., the *Initial Node* should be **TNodeStart** and the *Idle Action* property should be set to “Park at Home”). Select the “Transporting State” from the “Additional Symbols” and color it orange.

Notice that the **VEHICLE** object is similar to the **ENTITY** object in that multiple instances, which we call “run-time” instances, can be created from the “design-time” instance of the **VehInsideCart**.<sup>157</sup> “Design-time” occurs while you are putting instances of the objects onto the modeling. All the objects in the **FACILITY** window are initialized just before the simulation begins to execute. “Run-time” occurs when the simulation executes.

---

<sup>156</sup> The fact that the property description says “*Initial number in System*” should indicate that vehicles can be added and removed from the simulation during the simulation execution (something that is left to a later chapter).

<sup>157</sup> We will see that the **WORKER** object can also have run-time instances.

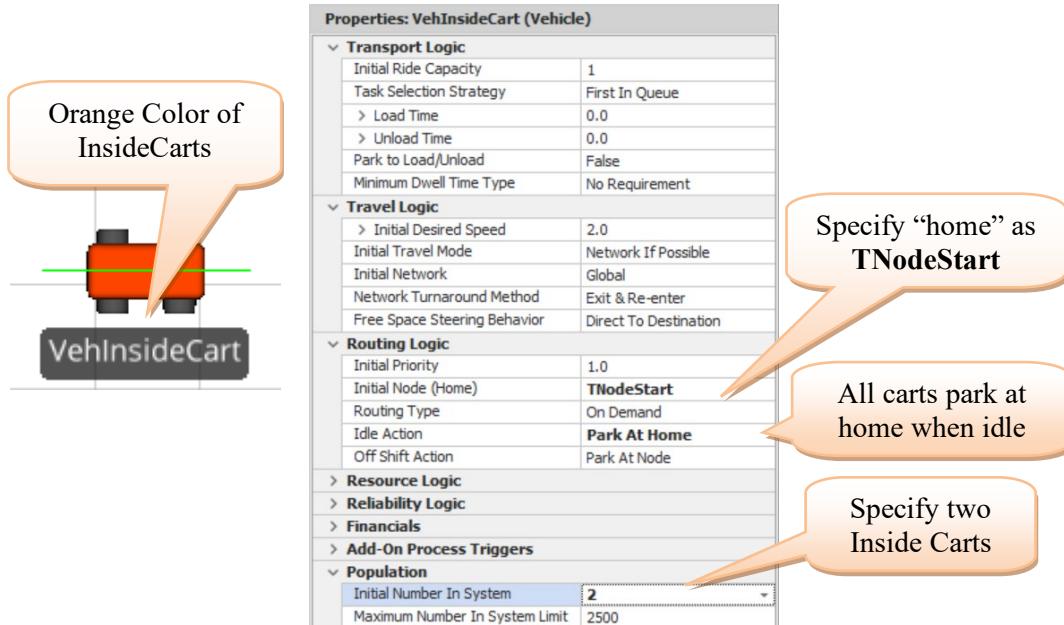


Figure 12.10: Properties of the InsideCart

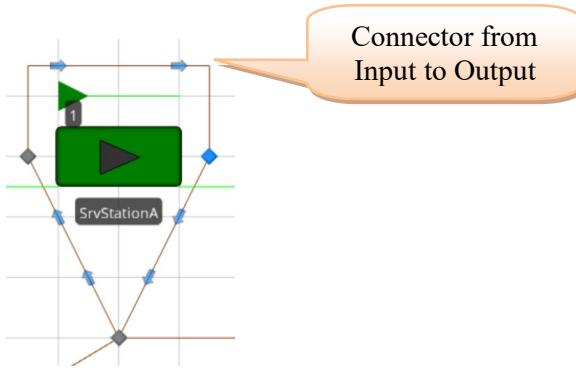
**Step 4:** We must ensure the inside carts can drop off parts and then return to the circular path. Since they obviously cannot go through the SERVERS<sup>158</sup>, the inside carts would move to the input node of a SERVER, drop off the part, and then go through free space to either the start node or to a pickup point rather than being on the network as discussed in Table 12.2. The easiest solution is to allow the carts to maneuver around the stations (i.e., add a CONNECTOR from the input to the output nodes of each station, as shown in Figure 12.11, making sure the direction of the link is correct.<sup>159</sup>

Table 12.2: Specifying the Initial Travel Mode

Initial Travel Mode	Description
Network if Possible (Default Method)	The transporter will traverse the network from A to B if there is a path between those nodes. Otherwise, it will go via free space (i.e., a straight line).
Free Space Only	The transporter will only use free space.
Network Only	The transporter will only use the network and could be deadlocked if there is no path from nodes A to B.

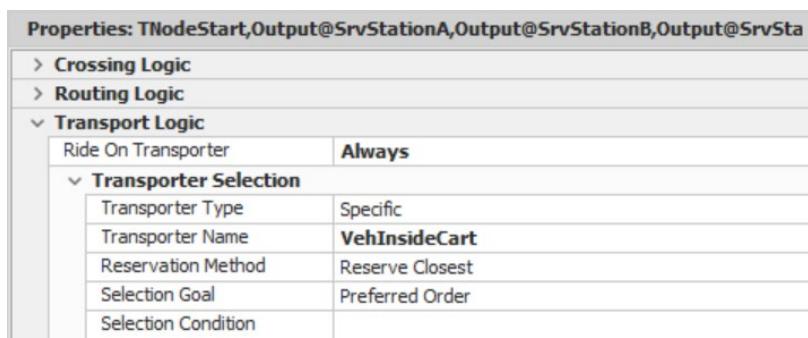
<sup>158</sup> By default, WORKERS and VEHICLES will not enter fixed objects like SERVERS, SINKS, COMBINERS, etc. One solution is to create a connector that leaves the input node to the transfer node, and another from the transfer node to the output node to allow pickup.

<sup>159</sup> Recall that connectors take up zero time and have zero length in the model. Basically, the server's inputs and outputs are really the transfer nodes of the circular path.



**Figure 12.11: Adding a Connector at a Station**

**Step 5:** Next, select the **TNodeStart** and the three output nodes (i.e., **TRANSFERNODES**) of the station servers and specify the entities leaving these nodes to ride on a **VehInsideCart**, as seen in Figure 12.12.

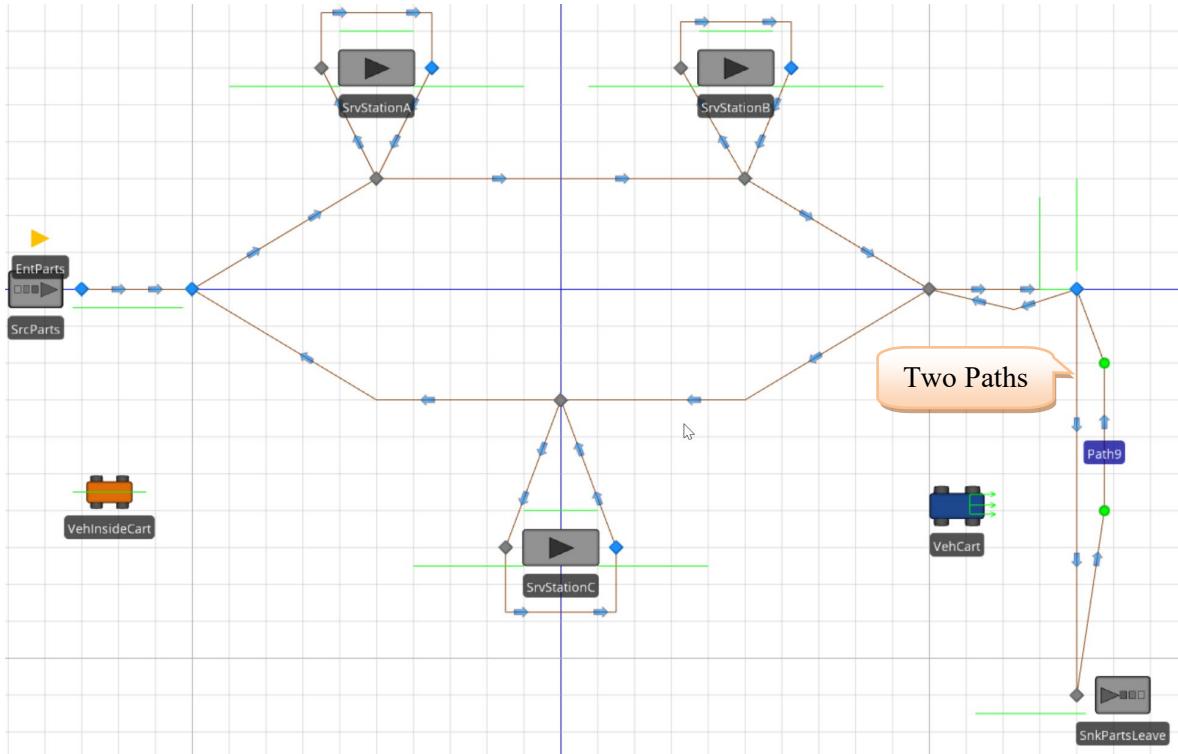


**Figure 12.12: Use the Inside Cart**

**Step 6:** Finally, change the path between the exit node on the inside loop and the **TNodeCart** node to two one-way paths (i.e., add an additional path from the **TNodeCart** back to the basic node, making sure to make the path ten yards long). Also, change the path between the **TNodeCart** and the **SnkPartsLeave** to two one-way paths. Now, your overall model should look like the one in Figure 12.13.<sup>160</sup>

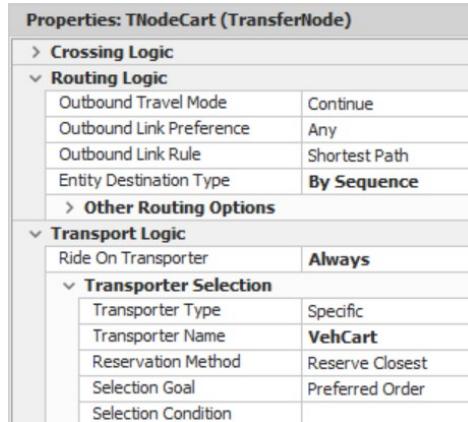
---

<sup>160</sup> One can use bidirectional paths but this can lead to bottlenecks and deadlock as vehicles, workers, and entities can only be going in one direction and must wait for all items to clear path before the direction can be reversed. It is always better to use two separate paths.



**Figure 12.13: Model with Vehicles**

**Step 7:** Also, change the **VehCart Population** property *Initial Number in the System* to “2” so two carts will be available for outside transportation. The **TNodeCart** node should also be specified, as shown in Figure 12.14.



**Figure 12.14: Specifying TNodeCart**

**Step 8:** Save and run the model, observing what happens.

**Question 4:** What did you notice about the utilization of the outside carts?

**Step 9:** The difficulty was the parts were not unloaded from the inside cart to be transported by one of the two outside carts to the **SINK**. The issue is that once the part has finished its last processing at a station, the next node in the sequence is the exit. Therefore, the inside cart takes part to its next destination (i.e., **Input@SnkPartsLeave**). To force the inside cart to drop off the parts at the **TNodeCart** node rather than

transporting them all way to the exit, this node should be inserted into the four-part sequences right before the **Input@SnkPartsLeave** row for each part type, as seen in Figure 12.15.<sup>161</sup>

Table Parts		Sequence Part		
	ID	Part Mix	Number to Arrive	Tally Stat Time In System
▶ 1	1	25	1	TallyStatP1
Sequence Part				
[x]	Sequence	Processing Times (Minutes)	ID	
1	Input@SrvStationA	Random.Pert(2,5,8)	1	
2	Input@SrvStationC	Random.Pert(2,6,11)	1	
3	TNodeCart	0.0	1	
▶ 4	Input@SnkPartsLeave	0	1	

**Figure 12.15: Adding the Transfer Cart to Force a Drop off of the Parts**

**Step 10:** Once all four sequences have been modified, save and execute the model. Observe the interacting behavior of the carts and notice how the carts park.

**Question 5:** How do parts get transported from the inside workstations to the outside cart for transfer to the sink when the parts leave?

---

**Question 6:** Is the queue ride parking for the **TransferCart** node needed anymore?

---

**Question 7:** What is the utilization of both outside **and inside Carts** at the end of the simulation (40 hours)?

---

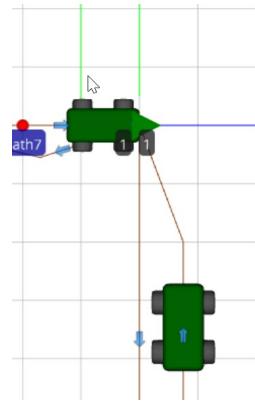
**Question 8:** How are the individual vehicles within a cart type distinguished in the output from each other?

---

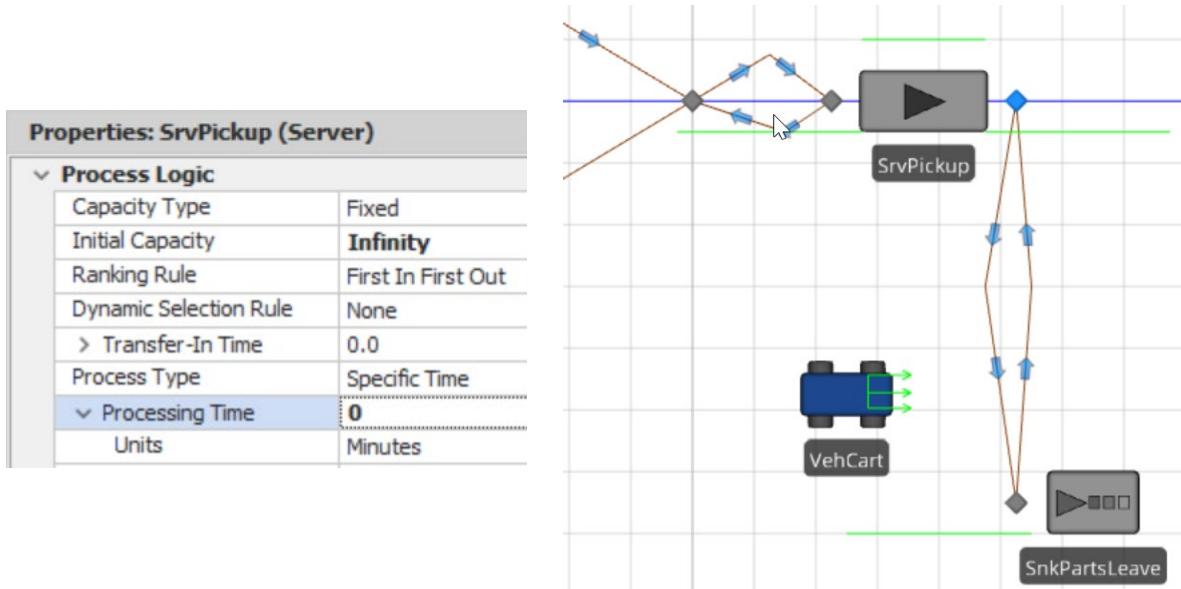
**Step 11:** A few anomalies occur within the model. Notice how the inside carts will wait while they drop off a part when an outside cart is unavailable. When the part starts the drop-off process, the part entity visits the node, informing the part that it requires a vehicle to continue and, therefore, never completes the drop-off process to release the inside cart as shown in To fix this anomaly and other animation issues, delete the **TNodeCart** node and insert a new SERVER named **SrvPickup** with an infinite capacity and zero for the processing time, as seen in Figure 12.17.

---

<sup>161</sup> The easiest way to insert the node into the sequence is to select the **Input@SnkPartsLeave** row and then click the *Insert Row* button in the *Data* section of the *Table* tab. Also, doing it part by part by accessing the related table from the Table Parts is easier.



**Figure 12.16: Inside Cart Stuck waiting for Entity to receive a Ride**



**Figure 12.17: Adding a new Server as Pickup Point**

**Step 12:** Reconnect the two 10-yard paths from the path network to the input of the **SrvPickup** and the two 300-yard paths from the output node to the **SINK**.

**Step 13:** For the sequence table, change the TransferCart node to the **Input@SrvPickup** and make the home node for the Cart the **Output@SrvPickup** node, as seen in Figure 12.18.

The screenshot shows two panels. On the left is a 'Sequence Table' with columns: ID, Part Mix, Number to Arrive, and Tally Stat Time In System. A row for '1' is selected, showing '25' in 'Number to Arrive' and 'TallyStatP1' in 'Tally Stat Time In System'. Below it is a 'Sequence Part' table with columns: Sequence, ProcessingTimes (Minutes), and ID. It contains four rows: 1 (Input@SrvStationA, Random.Pert(2,5,8), ID 1), 2 (Input@SrvStationC, Random.Pert(2,6,11), ID 1), 3 (Input@SrvPickup, 0.0, ID 1), and 4 (Input@SnkPartsLeave, 0, ID 1). On the right is a 'Properties: VehCart (Vehicle)' panel under 'Travel Logic'. It includes settings for Initial Desired Speed (4.0, Feet per Second), Initial Travel Mode (Network If Possible), Initial Network (Global), Network Turnaround Method (Reverse), Free Space Steering Behavior (Direct To Destination), and Routing Logic settings like Initial Priority (1.0), Initial Node (Output@SrvPickup), Routing Type (On Demand), Idle Action (Park At Home), and Off Shift Action (Park At Node).

ID	Part Mix	Number to Arrive	Tally Stat Time In System
1	25	1	TallyStatP1

Sequence Part			
	Sequence	ProcessingTimes (Minutes)	ID
1	Input@SrvStationA	Random.Pert(2,5,8)	1
2	Input@SrvStationC	Random.Pert(2,6,11)	1
3	Input@SrvPickup	0.0	1
4	Input@SnkPartsLeave	0	1

Properties: VehCart (Vehicle)	
> Transport Logic	
▼ Travel Logic	
Initial Desired Speed	<b>4.0</b>
Units	<b>Feet per Second</b>
Initial Travel Mode	Network If Possible
Initial Network	Global
Network Turnaround Met...	<b>Reverse</b>
Free Space Steering Beh...	Direct To Destination
▼ Routing Logic	
Initial Priority	1.0
Initial Node (Home)	<b>Output@SrvPickup</b>
Routing Type	On Demand
Idle Action	<b>Park At Home</b>
Off Shift Action	Park At Node

**Figure 12.18: Changing the Sequence Table and Cart Home Node**

**Step 14:** Finally, request transportation on the **Output@SrvPickup** transfer node using the same information from Figure 12.14.

**Step 15:** Save and run the model, observing what happens with the two types of carts.

**Question 9:** What is the utilization of both (outside) and inside carts at the end of the simulation (24 hours)?

### Part 12.3: Other Vehicle Travel Behaviors (Fixed Route and Free Space Travel)

Let's experiment with different vehicle specifications to create different cart behaviors. We will now explore both fixed-route travel (i.e., bus) and free space.

#### Fixed Route Travel

Currently, the inside cart travels wherever transport is needed among the stations, including entry and pickup based on the "On Demand" *Routing Type specified in the vehicle definition*. Suppose we want the cart to travel in a fixed sequence around all the service points in the model.

**Step 1:** Insert a new Sequence Table from the Data tab for the vehicle *Route Sequence named SeqInsideCart*, as shown in Figure 12.19. Note that the sequence includes both the stations' input and output nodes.

The figure shows two panels from the SIMIO software interface. On the left is a 'Seq Inside Cart' sequence table with 8 rows, listing actions: TNodeStart, Input@SrvStationA, Input@SrvStationA, Input@SrvStationB, Output@SrvStationB, Input@SrvPickup, Input@SrvStationC, and Output@SrvStationC. Row 8 is highlighted with a dashed border. On the right is a 'Properties: VehInsideCart (Vehicle)' panel showing the 'Routing Logic' section is expanded. It lists: Initial Priority (1.0), Initial Node (Home) (TNodeStart), Routing Type (Fixed Route), Route Sequence (SeqInsideCart), and Off Shift Action (Park At Node).

**Figure 12.19:** New Route Sequence for InsideCart

**Step 2:** Select the **VehInsideCart** and set the *Routing Type* property to “**Fixed Route**” and the *Route Sequence* to the **SeqInsideCart** sequence table, as seen in the right picture of Figure 12.19.

**Step 3:** Run the model and observe both the behavior of the cart(s) and the statistics on the cart’s performance.

*Question 10:* Are both inside carts behaving as expected? What is their utilization?

---

*Question 11:* Can you see the second cart? Why is its utilization 100%?

---

**Step 4:** Because both carts are always active, their utilization is 100%, and the two carts ride (i.e., overlap one another) in the same location. So if we are to see carts, we need to interrupt the constant motion or have the two carts start at different times (i.e., have a **SOURCE** create them at different times so there is space between them). There are several ways to achieve the result.

**Step 5:** Change the *Load Time* of the **VehInsideCart** to an **Exponential (0.5)** minutes, which will cause the vehicle to pause while loading each part. Save and run the model, observing the cart’s behavior.<sup>162</sup>

*Question 12:* What happens to both inside carts?

---

## Free Space Travel

Rather than travel on a specific path between stations, vehicles, workers, and entities can travel in “free space.” SIMIO needs the “location” of the various nodes to do this. In other words, it will assume that your model is “to scale,” as it might be if the animation layout is based on computer-aided drawing (CAD) software. It’s essential to recognize that the time it takes for parts to be moved by vehicles depends on the distance the vehicle travels and its travel speed<sup>163</sup>. To illustrate the free space travel, let’s treat our animation as though it came from a CAD drawing.

<sup>162</sup> We could have also added an unload time to simulate the time taken to unload parts at the various stations.

<sup>163</sup> When vehicles move entities, it’s the speed of the vehicle that determines how fast it traverses a distance and not the entity speed.

**Step 6:** Save the current model as Chap12.3Step6 and eliminate all the paths and nodes between the stations and the **SrvPickup** so that travel is unrestricted, as seen in Figure 12.20.

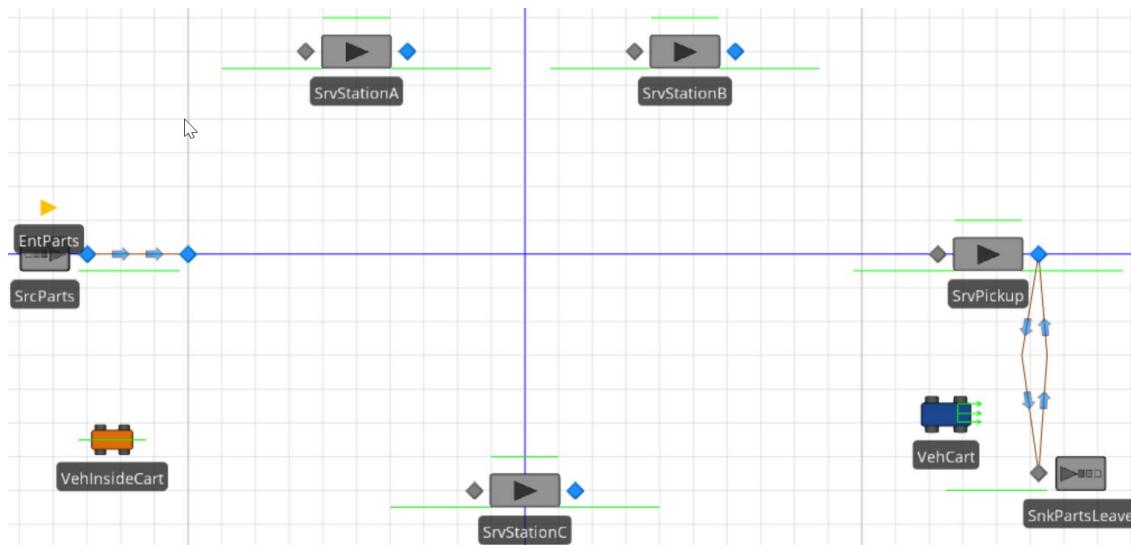


Figure 12.20: Layout to Scale: No Paths

**Step 7:** Modify the **VehInsideCart** “Travel Logic” and “Routing Logic” as Figure 12.21. In this case, we could change *Initial Travel Mode* to “**Free Space Only**” or leave it at the default “**Network if Possible**,” showing there is no travel network – meaning the vehicle must use free space. We are also routing “**On Demand**,” and the *Idle Action* is set to “**Remain Place**.”

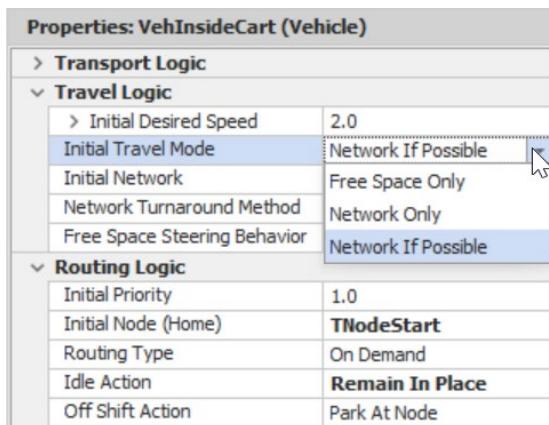


Figure 12.21: Travel Logic and Routing Logic

**Step 8:** Run the model and observe the travel of the inside carts.

**Question 13:** What do you observe about the parts when the source creates them?

**Step 9:** The part entity automatically floats through free space from the output node of the **SrcParts** to the first node in the sequence because the output node of the **SOURCE** says use by sequence to determine the part’s next destination. Since the default travel mode is “Network if Possible” and there are no direct paths from the **SOURCE** to the nodes, the part moves in free space. This travel will bypass the **TStartNode**, which specifies the entity’s need to ride on a transporter. Table 12.3 outlines two ways to fix the issue.

**Table 12.3: Forcing the Entities to Ride on the Transporter**

Object	Property	Description
<b>Output@SrcParts</b>	Ride on Transporter	Change the output node's "Ride on Transporter" property to " <b>Always</b> ," specifying the <b>VehInsideCart</b> as the transporter.
<b>EntParts</b>	Initial Travel Mode	Change the <i>Initial Travel Mode</i> of the entity to " <b>Network Only</b> ," forcing it to use the connector to the <b>TStartNode</b> .

*Question 14:* When would this Type of travel be most appropriate, and what do you observe about the parking?

---

#### Part 12.4: Conveyors: A Transfer Line

SIMIO provides for two kinds of conveyor modeling concepts: *accumulating* and *non-accumulating*. An accumulating conveyor allows parts to queue at its end, but the conveyor keeps moving. This approach might model a belt conveyor whose parts rest on "slip sheets" that allow the belt to continue to move. A non-accumulating conveyor must stop until the on-off operation is completed. This concept might apply to an overhead conveyor with parts on carriers. SIMIO conveyors do not have on/off stations, so a transfer line must be modeled as a series of individual conveyors. Since the transfer line stops, the non-accumulating concept applies. However, the individual conveyors will need to be synchronized.

Suppose a continuous conveyor transfer line handles the transfer among stations in the manufacturing cell, possibly in a loop with on/off stations. When a part is at an on/off station, the entire line pauses until the part has been placed on or taken off the conveyor. It takes between 0.5 and 1.5 minutes to load/unload the parts from the nodes.

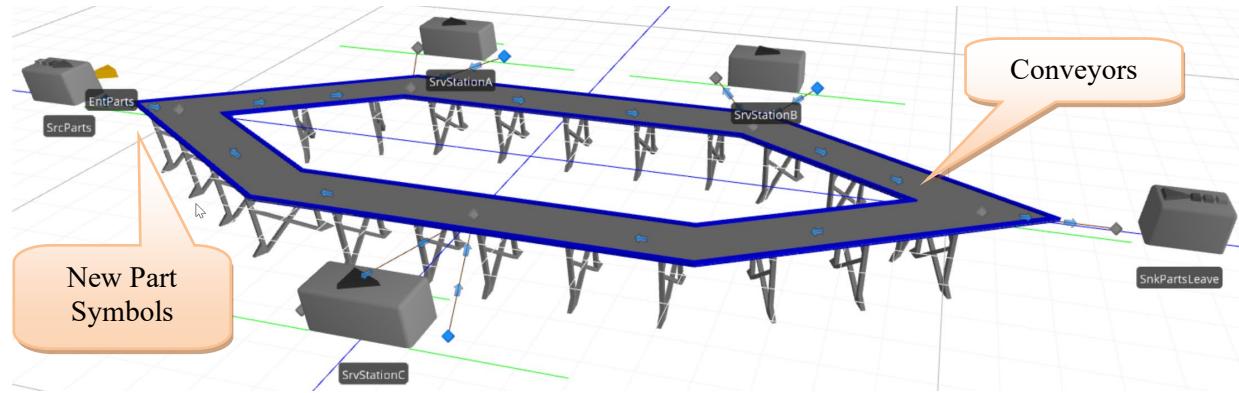
**Step 1:** Reopen the last model from Chapter 5 and save it with a new name. Now, change the arrival rate of the SOURCE to Random.Exponential(15) .

**Step 2:** Change all the paths on the interior of the model to conveyors by selecting the path and accessing the *Convert To Type* submenu via a right-mouse click.<sup>164</sup> Add path decorators for conveyors to embellish the model from the *Path Decorators* section of the *Edit* tab of the Link Tools, as seen in Figure 12.22.

**Step 3:** Also, the symbols for the parts should be modified so they are one meter wide by one meter in length and one-half meter in height. Select the **EntParts** MODELENTITY, and change the size under the *General→Physical Characteristics→Size* properties for the first active symbol. After selecting a different active symbol, you must deselect and reselect it to see its size. We will need to be a little more specific about these sizes as we size the conveyors. We created these symbols from the "Project Home" tab in the "Create" section, clicking on "New Symbol" and selecting "Create a New Symbol." when creating the symbol, we paid close attention to its size (the grid is in meters).

---

<sup>164</sup> Note you can use the *Ctrl* key to select all five paths and convert them all at once.



**Figure 12.22: 3D View with Conveyors**

**Step 4:** Next, the properties of the conveyors need to be specified. It is convenient to use the “Spreadsheet View” by right-clicking on one of the conveyors and selecting *Open Properties Spreadsheet*. The lengths from the previous paths (recall they were 25, 35, 20, 30, and 40 yards) do not need to be modified. All five conveyors should have a conveyor speed of 0.5 meters/second for each conveyor. Set each conveyor’s Accumulating property to False (i.e., the selection boxes should be unchecked).

**Step 5:** Choose the Cell Location option in the Entity Alignment property. Selecting this option causes the conveyor to be conceptually divided into cells. Parts going onto the conveyor will need to be placed into a cell. Therefore, choose the number of cells equal to the number associated with the conveyor length. Otherwise, if you select *Any Location* for the *Entity Alignment*, a part can go onto the conveyor anywhere, even interfering with another part (i.e., think about the baggage conveyor at an airport).

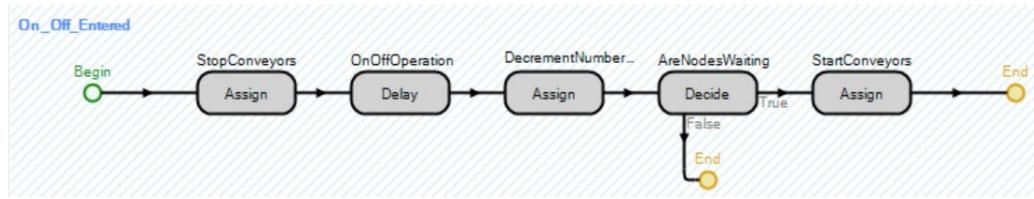
Travel Logic	Routing Logic	Reliability Logic	State Assignments	Financials	Add-On Process Triggers	Advanced Options	Advanced Options/Deprecated Properties	General	General/Physical Characteristics						
Instance Name	Initial Traveler Capaci...	Entry Ranking Rule	Entry Ranking Express...	Initial Desired Speed	Units	Entity Alignment	Cell Spacing Ty...	Number Of C...	Cell Size	Units	Auto Align Cells	Drawn To Scale	Logical Length	Units ▲	Accumulating
Conveyor1	Infinity	First In First Out	Entity.Priority	0.5	Meters per Seco...	Cell Location	Fixed Cell Size	1	1 Meters	With First Enti...	<input type="checkbox"/>	40	Yards	<input type="checkbox"/>	
Conveyor2	Infinity	First In First Out	Entity.Priority	0.5	Meters per Seco...	Cell Location	Fixed Cell Size	1	1 Meters	With First Enti...	<input type="checkbox"/>	25	Yards	<input type="checkbox"/>	
Conveyor3	Infinity	First In First Out	Entity.Priority	0.5	Meters per Seco...	Cell Location	Fixed Cell Size	1	1 Meters	With First Enti...	<input type="checkbox"/>	35	Yards	<input type="checkbox"/>	
Conveyor4	Infinity	First In First Out	Entity.Priority	0.5	Meters per Seco...	Cell Location	Fixed Cell Size	1	1 Meters	With First Enti...	<input type="checkbox"/>	20	Yards	<input type="checkbox"/>	
Conveyor5	Infinity	First In First Out	Entity.Priority	0.5	Meters per Seco...	Cell Location	Fixed Cell Size	1	1 Meters	With First Enti...	<input type="checkbox"/>	30	Yards	<input type="checkbox"/>	

**Figure 12.23: Specifying Properties for all Five Conveyors of the TransferLine (Spreadsheet View)**

**Step 6:** Since the transfer line consists of multiple conveyors rather than a continuous one, the conveyors have to be synchronized so that when parts enter an on/off location on the transfer line, the entire line, which is a set of conveyors, stops to await the on/off operation.<sup>165</sup> First, define a new DISCRETE STATE VARIABLE of type integer (from the *Definitions* tab) for the Model named **GStaNumberNodesWorking**. This variable will represent the number of on/off locations currently either putting parts onto or taking parts off the transfer line. When this state variable is zero, the transfer line is running, and when this variable is greater than one, the transfer line should be stopped.

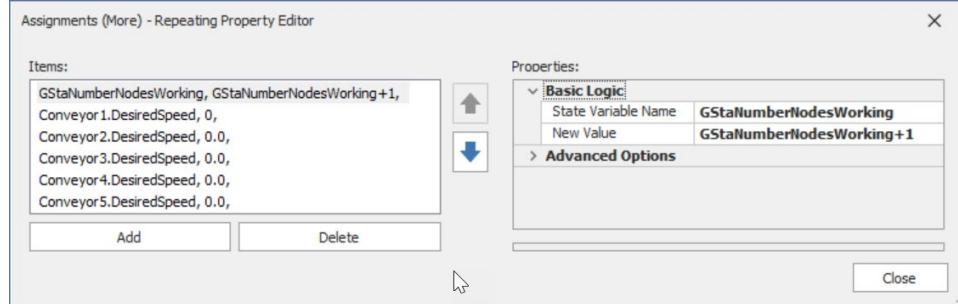
**Step 7:** Let’s now insert a new “process.” From the “Processes” tab, select “Create Process” and name the new process **On\_Off\_Entered**. The process will consist of the steps shown in Figure 12.24.

<sup>165</sup> The synchronization is not exact since the occupied cell of the upstream conveyor is not known to the downstream conveyor and parts from a cell may conflict with parts arriving from the upstream conveyor (i.e., **SrcParts**).



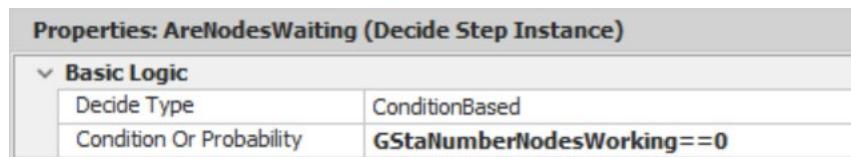
**Figure 12.24: Process when Entering/Leaving the Transfer Line**

- When a part reaches an on/off operation, the *Assign* step needs to increment the **GStaNumberNodesWorking** by one, and all conveyors need to be stopped by setting their desired speed to zero (see Figure 12.25).<sup>166</sup>



**Figure 12.25: Incrementing the Number of Nodes Working and Shutting Down the Conveyors.**

- The *Delay* step will be used to model the on/off time, which is assumed to be uniformly distributed with a minimum of 0.5 and a maximum of 1.5 minutes.
- The second *Assign* step will now decrement the **GStaNumberNodesWorking** by one.
- The *Decide* step is based on the condition that **GStaNumberNodesWorking==0**, meaning no other on/off stations are busy loading/unloading the transfer line.

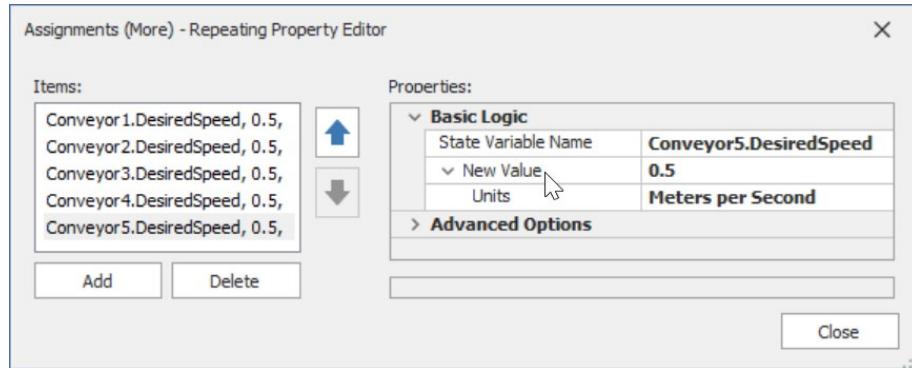


**Figure 12.26: Decide Step Option**

- The last *Assign* step will turn the conveyors back on. SIMIO stores all times internally in meters per hour. If the state variable has units, these can be specified in the assignment. Make sure you change the units to meters per second, or an assignment of 0.5 would be treated as 0.5 meters per hour.<sup>167</sup> You can convert the original specification of 0.5 meters per second to 1800 meters per hour to be consistent with how SIMIO will interpret the assignment and avoid having a conversion each time. Figure 12.27 shows the conveyors being turned back on.

<sup>166</sup> You will need to select the *Assignments (More)* button in the *Assign* step.

<sup>167</sup> The presumption of a standard internal time probably represents a choice of efficiency over ease of use, since determining the original time units would represent some additional computational effort.



**Figure 12.27: Turning the Conveyors Back On**

**Step 8:** Finally, the **On\_Off\_Entered** generic process must be invoked appropriately (i.e., every time a part is loaded or unloaded to the transfer line). Select the **On\_Off\_Entered** as the add-on process trigger for all the nodes and one path (i.e., the path to the exit **SnkPartsLeave**) as specified in Table 12.4.

**Table 12.4: All the Add-On Process Triggers that Need to Go Through the On/Off Process Logic**

Add-On Process Trigger	Which Object the Trigger is Associated
<i>Exited</i>	<b>Output@SrcParts</b> ( <b>TRANSERNODE</b> )
<i>Entered</i>	<b>Input@StationA</b> ( <b>BASICNODE</b> )
<i>Exited</i>	<b>Output@StationA</b> ( <b>TRANSERNODE</b> )
<i>Entered</i>	<b>Input@StationB</b> ( <b>BASICNODE</b> )
<i>Exited</i>	<b>Output@StationB</b> ( <b>TRANSERNODE</b> )
<i>Entered</i>	<b>Input@ SnkPartsLeave</b> ( <b>BASICNODE</b> )
<i>Entered</i>	<b>Input@StationC</b> ( <b>BASICNODE</b> )
<i>Exited</i>	<b>Output@StationC</b> ( <b>TRANSERNODE</b> )

**Question 15:** Why did we increment and decrement **GStaNumberNodesWorking** by one rather than just setting it to one or zero, which would eliminate the need for the **Decide** step?

---

**Question 16:** Why not simply use the **On\_Off\_Entered** process at the **BASICNODE** of the entry/exit?

---

**Step 9:** Run the simulation for 40 hours at a slow speed to ensure each part follows its sequence appropriately and answer the following questions.

**Question 17:** How long are all part types in the system (enter to leave)?

---

**Question 18:** What is the utilization of each of the **SERVERS**?

---

**Question 19:** What is the average number on the conveyor link between **SrcParts** and **SrvStationA**?

---

**Question 20:** What is the average time on the link between **SrvStationA** and **SrvStationB**?

---

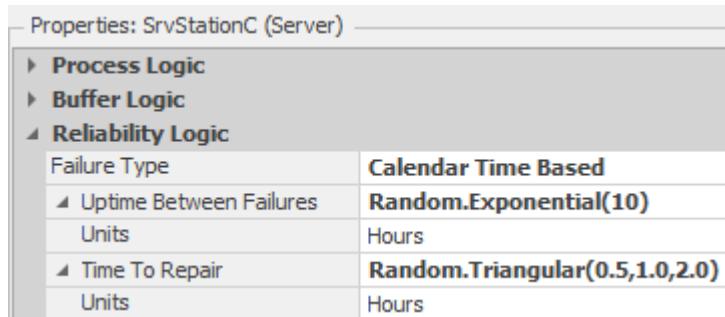
## Part 12.5: Machine Failures in the Cell

Failures are a part of most mechanical and other systems. SIMIO provides methods to model these failures within the “*Reliability Logic*” section of the SERVER (and other objects), as seen in Table 12.5. In the case of the transfer line, we also need to shut it down while a machine is being repaired.<sup>168</sup> Let’s assume that **SrvStationC** is an unreliable machine. In particular, we will assume that the MTBF (Mean Time Between Failures) is exponentially distributed with a mean of ten hours, while the MTTR (Mean Time To Repair) is modeled with a Triangular distribution (0.5, 1, 2) hours.

**Table 12.5: Reliability Failure Types**

Failure Type	Description
<i>No Failures</i>	Specify there will not be a failure.
<i>Calendar Time Based</i>	Specify the calendar time that needs to transpire before a failure will occur.
<i>Event Count Based</i>	Specify the number of times a particular event occurs that will cause the server to fail.
<i>Processing Count Based</i>	Specify the number of entities that are processed before a failure (e.g., after 100 entities, we need to change the tool).
<i>Processing Time Based</i>	Specify the total processing time that the server works before a failure (e.g., after 100 hours of work, we need to change the oil in the machine).

**Step 1:** In the “*Reliability Logic*” section for the Station C SERVER, specify the *Failure Type* to be “*Calendar Time Based*” and the *Uptime Behavior* to be *Random.Exponential(10)* with units of hours, and the *Time To Repair* should be *Random.Triangular(0.5,1,2)* with units of hours as seen in Figure 12.28. Remember that the server is failing, which means that all the capacity associated with this server fails (the capacity for this server is scheduled, but it changes between 1 and 2).

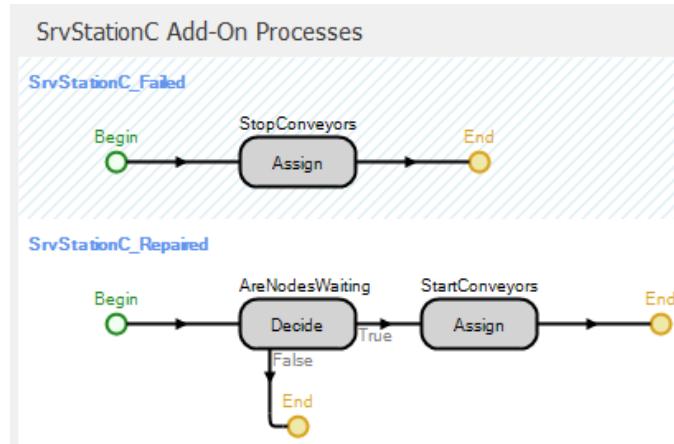


**Figure 12.28: Specifying a Failure and Repair**

**Step 2:** When the **SrvStationC** fails, the entire transfer line should be shut down and then started back up once the station has been repaired. Select **SrvStationC** and create new Add-On Process Triggers, “*Failed*” and “*Repaired*,” to produce new processes **SrvStationC\_Failed** and **SrvStationC\_Repaired**. The “*Failed*” trigger will be invoked at the start of the failure, while the “*Repaired*” trigger will be invoked after the SERVER has been repaired. In **SrvStationC\_Failed**, use an *Assign* step to turn off all of the conveyors by setting their DesiredSpeed to zero. In the **SrvStationC\_Repaired**, use an *Assign* step to turn back on the conveyors if there are currently no nodes loading/unloading (i.e., setting the DesiredSpeed = 0.5 meters per second for all conveyors) as seen in Figure 12.29.<sup>169</sup>

<sup>168</sup> The tremendous cost of downtime due to everything on the line becoming idle is one of the reasons why transfer lines and similar highly automated processes are becoming relatively rare.

<sup>169</sup> Copy the *Decide* and *Assign* steps from the **On\_Off\_Entered** process and remove the increment of the **GStaNumberNodesWorking**.



**Figure 12.29: Failed and Repairing Process Triggers**

**Step 3:** Also, the **On\_Off\_Entered** process should not start up the transfer line if the **SrvStationC** has failed and is currently being repaired. To do this, change the *Decide* step in the **On\_Off\_Entered** process so the conditional expression becomes the following.<sup>170</sup>

```
(GStaNumberNodesWorking==0) && (SrvStationC.Failure.Active==False)171
```

**Step 4:** Note that the **SrvStationC** will change to red when it is down, indicating that the server (and all its capacity) is in the “Failed” state. Run the model and ensure the transfer line stops when the station is also under repair.

*Question 21:* Is there anything about the final animation that troubles you?

---

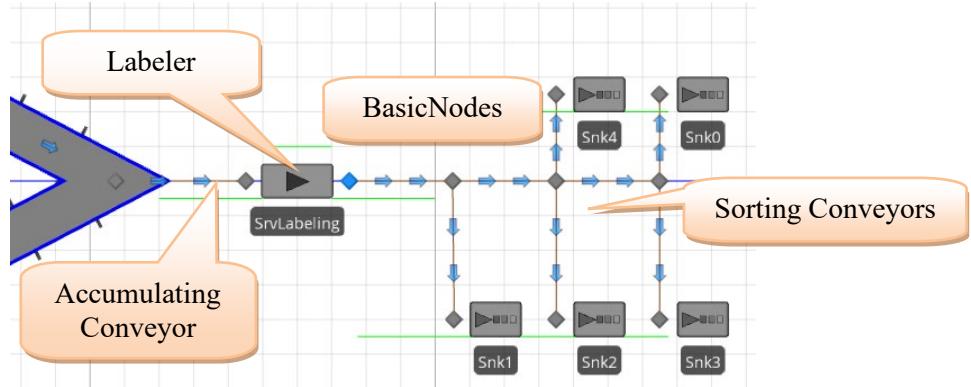
## Part 12.6: Sorting Conveyors

Now, we find out that the output from the conveyor system is another system of conveyors that sorts the parts into destinations. The parts that exit the manufacturing cell are routed to an accumulating conveyor at a labeling machine. From the labeling machine, the parts are conveyed to one of four sinks corresponding to the specific part type, as seen in Figure 12.30. The four conveyors directly in front of the sinks are accumulating conveyors. These conveyors keep moving, and parts are allowed to queue, although the sinks pose no barrier to exit in this case. The rest of the conveyors are non-accumulating.

---

<sup>170</sup> Object.Failure.Active function will return true if the current object has failed.

<sup>171</sup> Notice that SIMIO utilizes C# logical operators (i.e., == for equality, && for “And” statement, and || for the “Or” statement).



**Figure 12.30: Labeling and Sorting Conveyors**

**Step 5:** Remove the original **SnkPartsLeave** and replace it with a **SrvLabeling**. The labeling processing time follows a **Triangular(1,2,3)** minutes and has a capacity of “1.” Since the labeler will be supplied by an accumulating conveyor from the inside set of conveyors (i.e., the **BASICNODE**), set the *Input Buffer* capacity property to zero because it is not needed and there is no delay in off-loading to this conveyor.

**Step 6:** Add three **BASICNODES** to follow the **SrvLabeling** and name them **BNode1**, **BNode2**, and **BNode3** in that order. These are conveyor diverters that will send the sorted parts to their destinations. Add three sinks, **Snk1**, **Snk2**, **Snk3**, and **Snk0**, to represent the destinations for parts of type 1, 2, 3, and 0, respectively. Connect the objects with conveyors according to Table 12.6.

**Table 12.6: Conveyor Properties**

From/To	Conveyor Type	Conveyor Length	Conveyor Speed
BasicNode to SrvLabeling	Accumulating	10 meters	1.0 meters per second
SrvLabeling to Basic1	Non-Accumulating	10 meters	0.5 meters per second
BNode1to Snk1	Accumulating	10 meters	0.1 meters per second
BNode1to BNode2	Non-Accumulating	10 meters	0.5 meters per second
BNode2 to Snk2	Accumulating	10 meters	0.1 meters per second
BNode2 to Snk4	Accumulating	10 meters	0.1 meters per second
BNode2to BNode3	Non-Accumulating	10 meters	0.5 meters per second
BNode3to Snk3	Accumulating	10 meters	0.1 meters per second
BNode3to Snk0	Accumulating	10 meters	0.1 meters per second

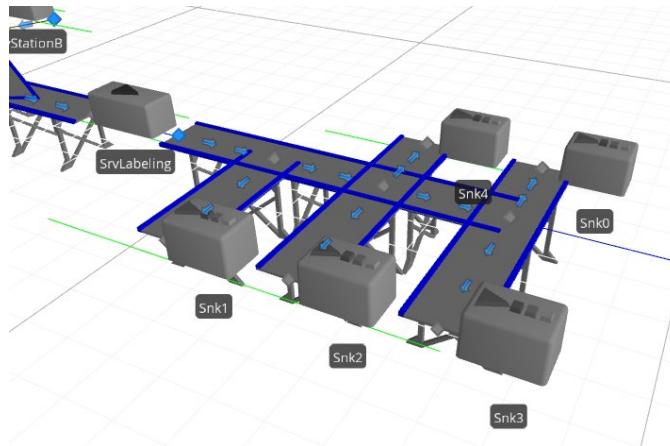
**Step 7:** For the part’s sequence table, replace the **Input@SnkPartsLeave** entries with **Input@SrvLabeling**. Next, add the sink node destinations associated with the part types. Figure 12.31 shows part of the table, and you can insert the last placement at the end to maintain the sequence.

14	Input@SrvStationB	Random.Pert(5,9,11)	5
15	Input@SrvStationC	Random.Triangular(6,9,11)	5
16	Input@SrvLabeling	0	5
17	Input@Snk0	0.0	1
18	Input@Snk1	0.0	2
19	Input@Snk2	0.0	3
20	Input@Snk3	0.0	4
21	Input@Snk4	0.0	5

**Figure 12.31: Changes to the Sequence Table**

Next, change the **Output@SrvLabeling** node **Entity Destination Type** to **By Sequence**. There is no need to add intermediate nodes (the **BASICNODES**) since SIMIO can find the designations from the **SrvLabeling**.

**Step 8:** Select all the conveyors and add the conveyor path decorators, as seen in Figure 12.32.



**Figure 12.32: Conveyor with Path Decorators**

**Step 9:** Save and run the simulation while observing the animation, which may need to be slowed down.

**Question 22:** Are the parts being conveyed as expected, and do you see accumulation at the labeler?

---

**Step 10:** Run the simulation for 40 hours.

**Question 23:** What is the average number of parts accumulating on the conveyor before the labeler?

---

**Question 24:** What is the utilization of the labeler?

---

**Question 25:** Why does the OUTPUTBUFFER of the labeler have content?

---

**Question 26:** What is the average number on the conveyor between **BNode2** and **BNode3**?

---

**Question 27:** Do you have any concerns about the operation of the sorting conveyor system?

---

## Part 12.7: Commentary

- Material handling can be a substantial cost of production and thus is of great interest to many companies. Vehicles and conveyors provide considerable flexibility in modeling material handling.
- The modeling of cranes has not been included in this chapter. These are a complex category of materials handling. However, SIMIO has developed a special library for cranes that can represent a wide variety of overhead and floor cranes. In particular, there is a 3D animation of the cranes to reflect that cranes operate in three dimensions. This package is highly recommended.

# Chapter 13

## Management of Resources: Veterinary Clinic

---

The RESOURCE object provides a flexible means to model individual resource needs. Unlike the SERVER, the RESOURCE does not provide a processing capability with input and output buffers at a specific location. Further, because the RESOURCE object is not location-restricted, the units of its capacity can be used at several places within a model. Because the resource is flexible, its capacity can be allocated (i.e., seized and released) in a complex fashion. The management of a firm's resources is one of the main challenges of any organization. A simulation of that management can greatly assist in determining how those resources can be best used.

To illustrate part of the resource management capability, a Veterinary Clinic is modeled. The veterinary clinic provides care for many different kinds of animals. The office wants to provide efficient care through the employment of efficient resources. For simplicity, we primarily model the activities of the veterinary staff. The staff consists of four veterinarians named (first names) Billy, Jim, Maisy, and Ellie. Several factors complicate this clinic service. There are times when multiple types of resources, such as a Veterinary staff member and an X-ray machine, are required before the patient is serviced. There are certain preferences for resources, such as Jim being the only veterinarian who can service iguanas. The SERVER object cannot handle these cases easily. For instance, you may need a nurse and doctor to perform a procedure and then need the nurse for some after-procedure service.

### Part 13.1: Utilizing the Fixed Resource Object

The RESOURCE object is a capacitated “Fixed Object” (i.e., it does not visibly move in the network similarly to the TRANSFERNODE or SOURCE objects) and can be seized and released. It does provide a generic object that can be used to constrain entity flow. RESOURCES have reliability capabilities and the ability to statically and dynamically rank and select the entities for processes (i.e., process all critically ill patients first). They can also choose to remain idle under some conditions. (i.e., refuse to be seized).

**Step 1:** Create a new fixed model (“Project Home” tab). Note that you can have multiple models in the same project, but only one will run at a given time unless you use them “inside” each other, as seen in Figure 13.2.

**Step 2:** Add a new SOURCE named **SrcPatients**, a SERVER named **SrvVetClinic**, and a SINK named **SnkExit**. Add a MODELENTITY named **EntPatients**. Connect the objects with paths that are logically 30 meters long.<sup>172</sup>

- Patients arrive Exponentially with an interarrival time of 6.5 minutes.
- Each patient takes between 10 and 30 minutes to be seen by the staff, with most patients taking 15 minutes to be served (i.e., use a Pert distribution to model the processing time).

**Step 3:** From the Standard Library section, add a new fixed RESOURCE<sup>173</sup> named **ResBilly**.

---

<sup>172</sup> Recall, if you select the output node of the SOURCE while holding down the *Shift* and *Ctrl* keys, SIMIO will allow you draw LINKS without first selecting a LINK type from the Standard Library. This key sequence can be more efficient. Once the two PATHS have been drawn, select them both and change their *Drawn to Scale* properties to “False” and the *Logical Length* properties to 30 meters.

<sup>173</sup> The default symbol looks like a gray eye with a blue iris.

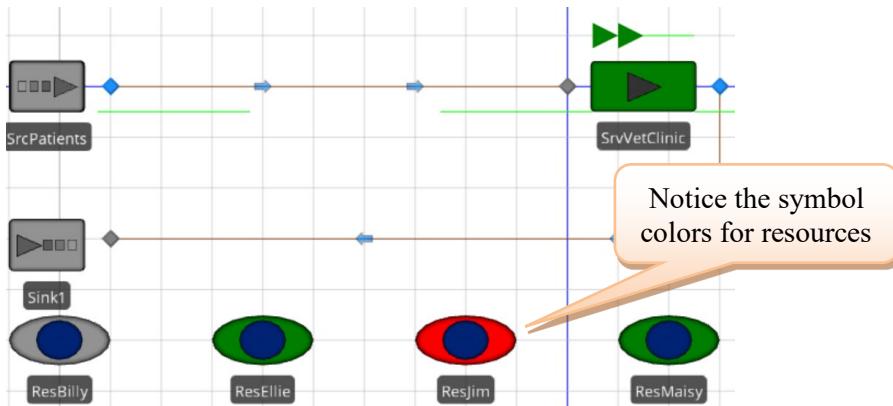
- Since RESOURCES have reliability logic, we can use it to model emergency phone calls for the vets. Specify the *Uptime Between Failures* to follow an exponential distribution with a mean of six hours and *Time to Repair* expression equal to a Random.Pert(40, 55, 60) minutes, as shown in Figure 13.1.



**Figure 13.1: Reliability Logic to Handle Emergency Calls from the Veterinarian**

- Like the servers, the resource symbol changes as its state changes. If you click on **ResBilly** and examine the “active symbol,” there are seven resource states. The first five states are the same as those for servers, but there are two additional states (i.e., “Failed Busy” and “Off Shift Busy”). The symbols change colors, while the iris remains blue (i.e., the color is red when a resource fails). Similar to the server states (see Chapter 8), the resource states have a name, color, and state number.
- Like the servers, the resource is idle (state 0) when all the capacity of the resource is idle, is busy when any of the capacity is being used, blocked whenever the resource is presented from disengaging from an entity, failed whenever the reliability causes a failure of the resource, and offshift when the capacity of the resource is zero.

**Step 4:** Copy the **ResBilly** resource instance three times, naming them **ResEllie**, **ResJim**, and **ResMaisy**



**Figure 13.2: Model of the Veterinary Clinic Using Four Resources**

**Step 5:** Change the capacity of the SERVER **SrvVetClinic** to *Infinity* so the SERVER’s capacity does not constrain the system. We want the new resource objects to be the constraining requirement.

**Step 6:** At this point, we can seize an individual RESOURCE, but our model will require selecting among any available staff member rather than a specific resource. Go to the “Definitions” tab of the model and create a new LIST of OBJECTS named **ListResources**. Next, add all four resources to the list, as seen in Figure 13.3. Patients will use this list to select an available veterinary staff.

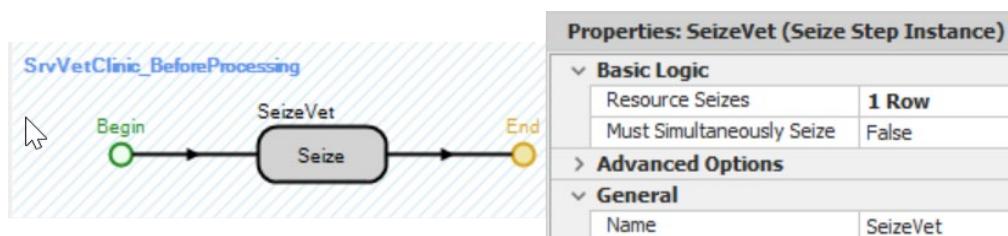
Objects	
	Object
▶	ResBilly
	ResEllie
	ResJim
	ResMaisy

**Figure 13.3: Creating a List of Resources to be Seized**

**Step 7:** The basic model has been set up with fixed RESOURCES; however, the patients need to request (seize) an available staff member to be serviced and then, once they have completed service, need to release the staff member to work on the next patient. Patients will request a staff member when they arrive at the Vet Clinic. Two *Add-On Process Triggers* for the SERVER will be used.

**Step 8:** The patient entity needs to seize one of the available resources before entering processing (i.e., the “Before Processing”<sup>174</sup> trigger). To invoke the “Before Processing” add-on process, expand the “Add-On Process Triggers” in the properties window and double-click on the label to invoke/create a process.<sup>175</sup>

**Step 9:** Since the patient wants to seize the resource, insert a *Seize* step by selecting from the list of possible steps and dragging it between the *Begin* and *End* nodes, representing where the process starts and ends, as seen in Figure 13.4.



**Figure 13.4: Using a Seize Step to Request a Veterinary Staff Member**

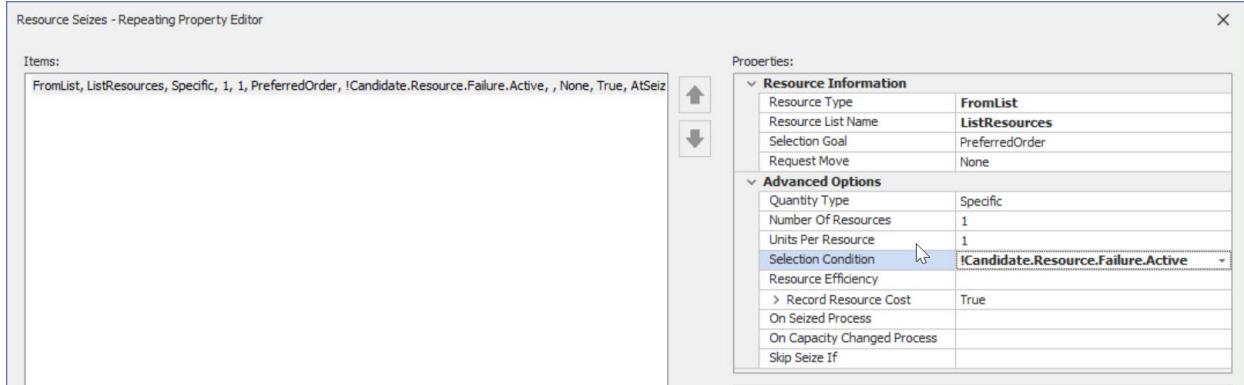
- Under the “General” properties of the process step, as seen on the right side of Figure 13.4, set the *Name* to **SeizeVet**. The resources to try to seize will be specified under the “Basic Logic.” Click on the button in the property window in the *Seizes* property row, which currently indicates that one type of resource has been specified as being seized.
- The *Seize – Repeating Property Editor* (see Figure 13.5) will pop up and allow you to specify as many resources that need to be seized before the entity can be processed. In this example, we will specify one resource type, and the *Object Type* will be “FromList”<sup>176</sup> which can be selected from any of the resources in a list (i.e., **ListResources**).

<sup>174</sup> The *Before Processing* add-on process trigger is invoked after the entity has seized capacity of the server but before the entity is physically transferred to the processing station while the *Processing* add-on trigger is invoked after the entity has been transferred and directly before the delay for processing. The *After Processing* add-on process trigger is invoked directly after the processing has finished.

<sup>175</sup> Note that we can utilize the secondary resources to seize and release additional resources (i.e., *Resource for Processing*) using the same list. However, utilizing processes gives you more flexibility and control.

<sup>176</sup> The *Object Type* for the seize can be “Specific”, “From List”, or “Parent Object”. The “Specific” option allows you to select an individual RESOURCE (e.g., **ResJim**), any object that has capacity (e.g., Servers, Paths), or any of the Stations inside objects that contain

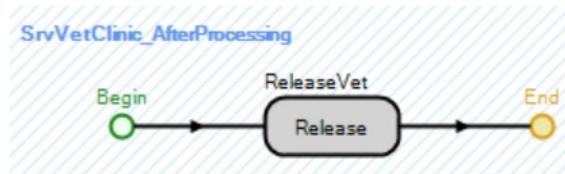
- We do not want to select a resource that is currently failing, so the *Selection Condition* property has to evaluate to true for each resource in the list for the resource to be considered as a possible choice to be seized. Therefore, set the property to `!Candidate.Resource.Failure.Active`.<sup>177</sup>



**Figure 13.5: Seize Property Editor**

**Step 10:** The patients are now able to *Seize* the clinic staff, but once the patient has finished processing, the entity (patients) will need to release the staff members (resource). There are two potential triggers (i.e., “*After Processing*” or “*Exited*”). Insert a new “*After Processing*” add-on process trigger by double-clicking on the label to invoke/create a process that will be executed once the process has been completed before exiting.

- Insert a *Release* step into the process between the *Begin* and *End* nodes, allowing the patient to release the resource seized, as seen in Figure 13.6.



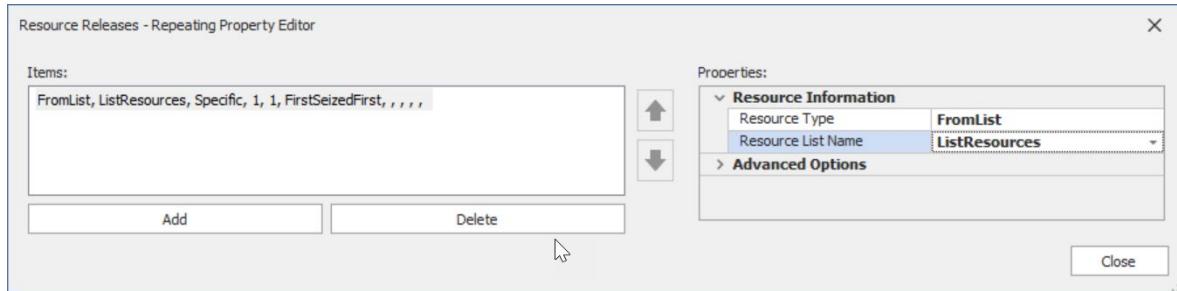
**Figure 13.6: Using a Release Step to Release a Veterinary Staff Member**

- Repeat a similar process to the *Seize* step, naming the step **ReleaseStaff** and specifying the *Releases* property. Identical to the Seize Editor, the *Releases* → *Repeating Property Editor* will allow one to release resources according to the release properties.

---

the capacity. The “Parent Object” will allow you to seize capacity of the object that is running the process which for the SERVER in this process is the **Processing STATION**. The list option will specify any list that contains objects that can be seized.

<sup>177</sup> The CANDIDATE object is a wild card that will be replaced with each RESOURCE in the list that is currently not seized.



**Figure 13.7: Release Property Editor for the Release Step**

**Step 11:** Save and run the model for 40 hours and observe what happens.

*Question 1:* What is the overall average utilization for each of the RESOURCES, and why does Billy have the highest utilization?

---

*Question 2:* What is the total time a patient has to wait for a staff person, and what is the average number in the queue for the clinic's input buffer?

---

*Question 3:* What is the total cycle time for the patients?

---

## Part 13.2: Different Resource Needs Based on Different Patient Types

The previous model assumed all clinic vets could see all patients. However, suppose Billy and Maisy have expertise with birds while Jim has expertise with iguanas, but if he is busy, Ellie can provide the service. Horses and cows are troublesome animals owing to their size and require two veterinarians. All other animals (cats and dogs) can be seen by any one of the veterinarians.

**Step 1:** Create a new data table named **TableAnimals** that has a “String” column(property) for the animal name, an “Integer” column for the animal type, and a “Real” column for the percentage of time this type of animal typically arrives to the clinic. The entry values are shown in Table 13.1. For example, 25 percent of the time, large animals representing cows and horses will arrive at the clinic.

**Table 13.1: Table with Animal Information and Types**

Table Animals			
	Animal	Type	Percent Type
1	CatsDogs	0	47
2	CowsHours	1	25
3	Birds	2	20
4	Iguanas	3	8

**Step 2:** Create two new OBJECT lists named **ListBird** and **ListIguana** in a similar manner to **ListResources** in Figure 13.3, which represents the staff that can service these specialty animals as seen in Figure 13.8.

Object	Properties: ListBird (ObjectList Instance)		Object	Properties: ListIguana (ObjectList Instance)	
	General			General	
ResBilly	Name	ListBird	ResJim	Name	ListIguana
ResMaisy	Description		ResEllie	Description	

Figure 13.8: Object Lists for the ListBird and ListIguana

**Step 3:** Add three additional symbols to the patient entity to distinguish between the various animal types. One can use four different colors, but you can also use interesting pictures of these creatures by downloading symbols from Trimble 3D Warehouse, as seen in Figure 13.9, which shows the four symbols horizontally (instead of vertically). Note that the order of the symbols should correspond to the type column Table 13.1.<sup>178</sup>

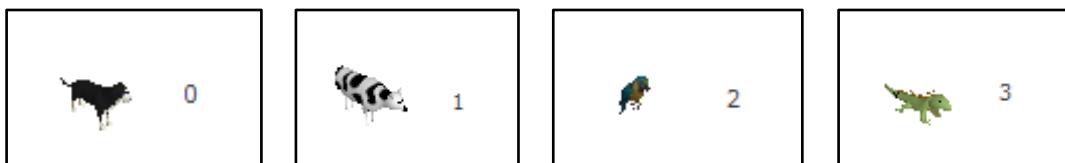


Figure 13.9: Trimble 3-D Warehouse Downloaded Dog, Cow, Bird, and Iguana Symbols

**Step 4:** The goal will be to randomly assign the animal types and pictures (color) based on the historical percentages. The assignment must be done after the entity has been created using the “Created” add-on process trigger in the SOURCE. Insert a new “Created” Add-On Process Trigger<sup>179</sup>, which will be executed once the entity has been created by the **SrcPatients**.

- One of the features of SIMIO is the ability to associate one or more tables and/or rows of a table with an entity. Our newly created patient needs to be associated with a specific row of the **TableAnimals** data table, and the correct picture symbol must be assigned based on the “Animal Type” column. Insert a *Set Row* and an *Assign* process steps into the process, as in Figure 13.10.<sup>180</sup>

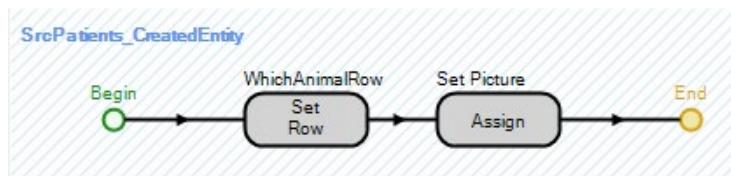


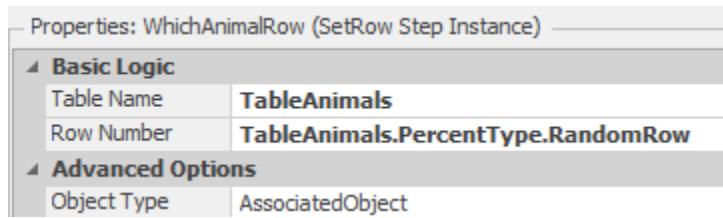
Figure 13.10: Determining Patient Type by Setting a Row of Datatable to Entity

<sup>178</sup> Recall that you need to download all four animal pictures as Sketchup™ files. Then, select each symbol in the Entity, and import the downloaded symbol. You may need to orient or resize the symbol based on the forward direction.

<sup>179</sup> To invoke the “Created” add-on process, expand the “Add-On Process Triggers” of the SOURCE in the properties window and double-click the label to invoke/create a process.

<sup>180</sup> In this example, the *Table Reference Assignments* (i.e., *On Created Entity*) could be used to assign a table. Without knowledge of the object, you cannot be sure that the picture assignment via *State Assignments* happens before or after the table assignment, but with the processes approach you can control the order.

- Specify a specific row of the associated table randomly using the `RandomRow` method based on the “PercentType” column, as seen in Figure 13.11.<sup>181</sup> Also make sure the *Object Type* under the *Advanced Options* is set to the “AssociatedObject”.<sup>182</sup>



**Figure 13.11: Specifying the Properties of the Set Row Process Step**

- Once an animal type has been determined, Figure 13.12 shows the properties of the *Assign* process step used to assign the associated “Type” column from the table to the picture symbol state of the `MODELENTITY`.<sup>183</sup>



**Figure 13.12: Changing the Picture of Entity to Reflect a Column in an Assigned Datatable**

**Step 5:** Save and run the model to ensure the different animal types arrive as expected. You may need to adjust the animated processing queues to allow the large animals to be visible.

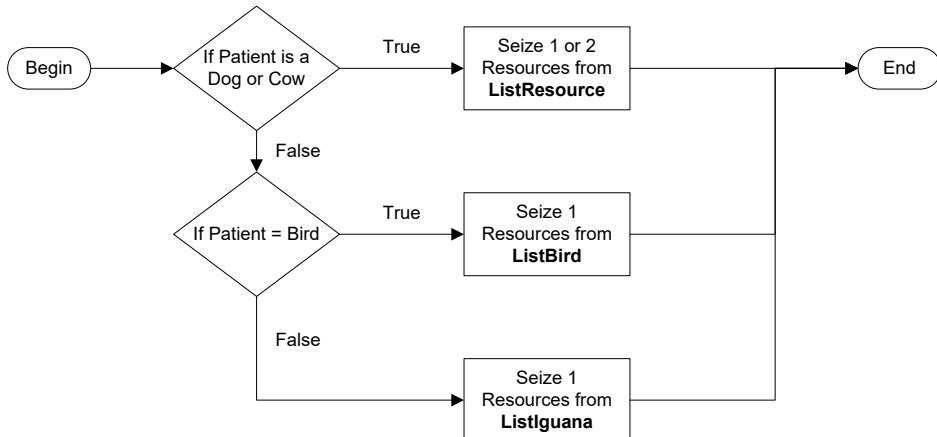
**Step 6:** The patients still seize any vet regardless of animal type, and only one staff member is seized. Therefore, we need to modify the Add-On Process trigger for seizing and releasing resources to incorporate the resource requirements (See flow chart in Figure 13.13),

- If the type is cat/dog or horse/cow, we can choose any resource, but for horse/cow patients, two resources are needed.
- If the type is a bird, we need to choose a resource from the OBJECT LIST **ListBird**.
- Finally, if the type is an iguana, we will need to choose from the OBJECT LIST **ListIguana**.

<sup>181</sup> Any numeric column can be used to specify a random row of a table. The numbers do not need to add up to 1 or 100. SIMIO will take the sum of the column and then generate a random row based on the normalized percentages.

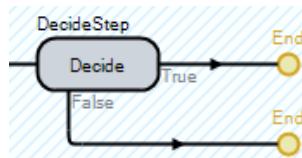
<sup>182</sup> The “ASSOCIATED OBJECT” is the object the current token represents. In this case, it is the entity or patient. The other options are “TOKEN” for the current token executing the process, the “PARENTOBJECT,” which would be the SOURCE since it owns the Add-on Process Trigger, and “SpecificObjectOrElement” which allows you to set the table on a particular object or element.

<sup>183</sup> Notice, the animal types were specified on purpose to be zero to three to represent the symbol numbering for the pictures. Also, the green arrow lets the user know this refers to a referenced property.



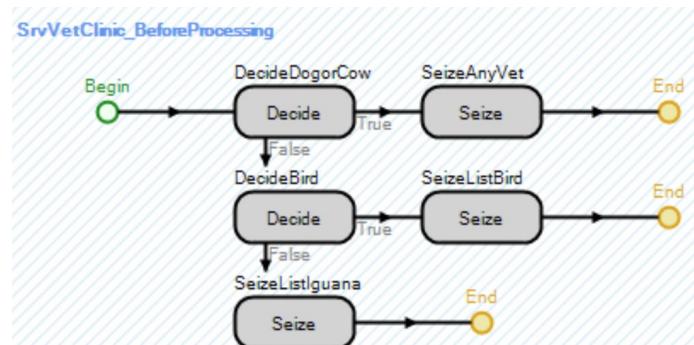
**Figure 13.13: Seizing the Appropriate Veterinary Staff Based on Patient Type**

- We need to make some decisions based on the animal type to perform the logic, as seen in the above flow chart. The logic is performed by a SIMIO *Decide* process step (see Figure 13.14), which is basically an If/Else statement with a **True** and **False** branch. In Figure 13.13, the *Decide* step's two major properties of interest are the *Decide Type* and the decision *Expression*.



**Figure 13.14: SIMIO Decide Process Step**

Figure 13.15 shows the modified “*Before Processing*” add-on process trigger that will seize the appropriate staff members based on animal type.<sup>184</sup>



**Figure 13.15: Seizing the Appropriate Resource Based on Animal Type**

- Insert a *Decide* step before the current *Seize* step, which will determine if the patient is a dog/cat or cow/horse. Based on which is true, the patient will seize from the resource object **LIST ListResource** as

---

<sup>184</sup> The *Processing* add-on process trigger occurs after the entities are transferred into the processing station so the animals will appear in the processing station even though they have not seized a vet. Therefore, use the *Before Processing* add-on process trigger to eliminate the animation issue.

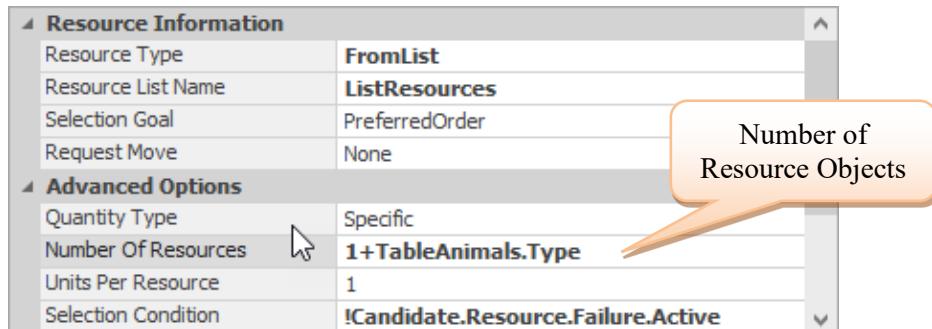
before. Specify the *Decide Type* to be condition-based with an expression that checks to see if the animal type is a 0 or 1 (see Figure 13.16).<sup>185</sup>



**Figure 13.16: Specifying the ConditionBased Expression of a Decide Step**

- The second *Decide* step will check whether it is a bird. (Hint: copy the first *Decide* block, paste it onto the False branch, and change the expression to (*TableAnimals.Type==2*)).
- Copy the *Seize* step to both the *True* and *False* branches of the second *Decide* step. You must change which resource list the entity will seize from (i.e., **ListBird** and **ListIguana**, respectively).
- The first *Seize* step is already set up to seize from all the resources (**ListResources**). However, by default, it only seizes one object. However, two resources are needed for large animals. There are several ways to model the logic. Since the animal type of cats/dogs is zero and cows/horses is one, the type can be added to one to give a result of one or two. In the *Number of Objects*<sup>186</sup> property to seize, the following Figure 13.17 illustrates the expression of seizing one or two objects based on the animal type.

**Step 7:** Change the **SrvVetClinic**'s processing animated queue alignment to “*Oriented Point*.” Then, add two more vertexes and orient all four toward the **SERVER**, as seen in Figure 13.18

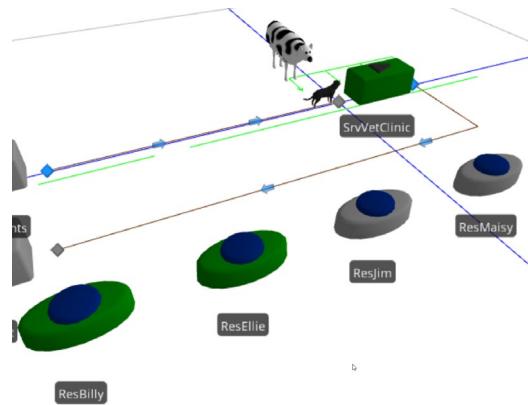


**Figure 13.17: Specifying an Expression to Determine the Number of Resources to Seize**

**Step 8:** Save and Run your model. You may need to experiment with the percentages to ensure the correct staff members are servicing the animals. Figure 13.18 shows a cow being serviced by both **ResBilly** and **ResEllie**.

<sup>185</sup> The expression (*TableAnimals.Type < 2*) determines if the type is a zero or one could have been specified (*TableAnimals.Type == 0 || TableAnimals.Type ==1*) where **||** equals the *Or* logical operator.

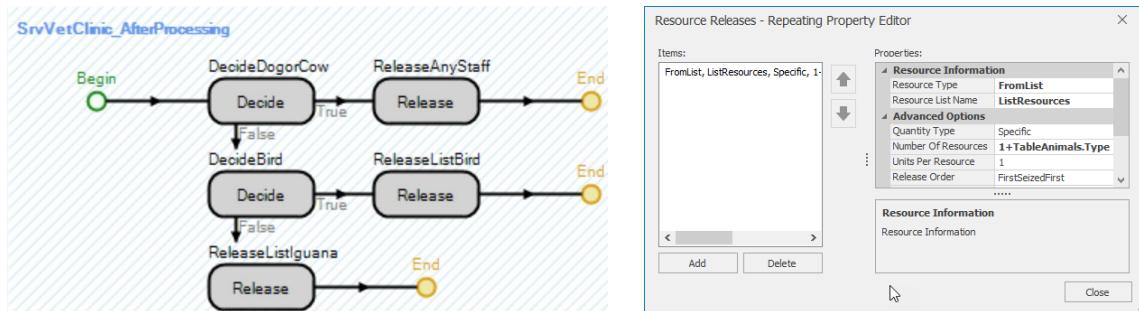
<sup>186</sup> The *Number of Objects* property specifies the unique number of objects to select out of a list while the *Units Per Object* specifies the number of each object to seize. For example, if your list had doctors and nurses and you set the *Number of Objects* to “1” and *Units Per Object* “2”, then it would seize either a doctor or nurse object but it would require two doctors or two nurses. Resources can be given capacity of more than one (e.g., you could have ten nurses but you do not want to have ten different nurses (i.e., Billy, Ellie, etc.).



**Figure 13.18: Using Two Resources to Service a Large Animal.**

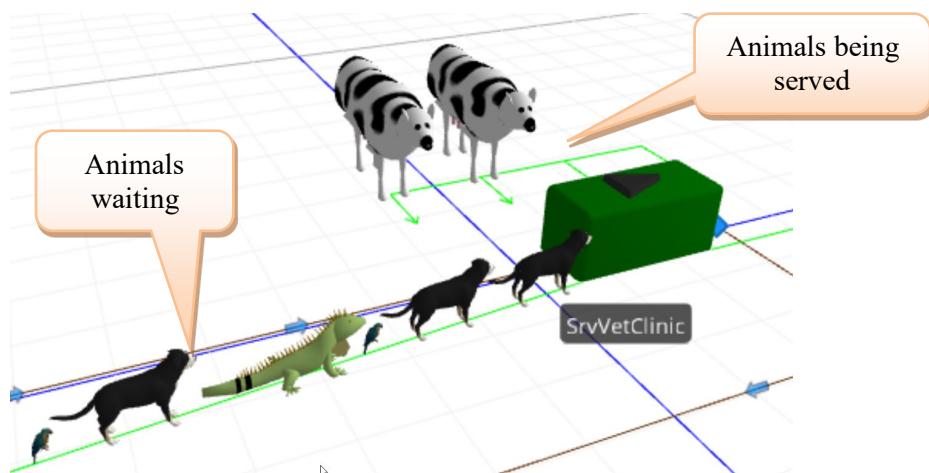
*Question 4:* Did the staff get released, and why?

**Step 9:** The only thing left to do is to release the veterinary staff once they have finished processing the patient. It is similar to the “*Before Processing*” trigger but uses *Release* instead of *Seize* steps. It is important that you release the correct number of objects back to the **ListResources** pool.



**Figure 13.19: Releasing the Correct Vet to Appropriate Resource Set**

**Step 10:** Save and Run your model and observe what happens (see Figure 13.20).



**Figure 13.20: Figuring Showing Animals being Serviced and Waiting**

*Question 5:* What is the overall average utilization of each of the RESOURCES?

---

*Question 6:* What is the total time a patient has to wait for a staff member and the average number in the queue for the clinic's input buffer?

---

*Question 7:* What is the total cycle time for the patient?

---

### Part 13.3: Resource Decision Making

One of the most prominent features of SIMIO is that you can model very complex logic (e.g., process steps) or expressions and supplement it by associating information via data tables on entities. In an earlier chapter, the processing time and routing information for different part types were linked directly to the part itself. When parts arrived at the SERVERS, the server would ask the entity how long it took to process them. Many simulation languages, including SIMIO, have the ability to specify attributes (i.e., entity state variables in SIMIO), which can be assigned the actual processing time value when the entity is created. However, SIMIO information in a data table can be dynamically changed for any object to reference it. In the current example, there are four different animal types. What if the number of types increases to ten with eight or nine different lists to seize the correct vet? The decision logic would get very complicated and long. Therefore, we have the ability to make the objects more intelligent (i.e., they know the list and the number required for service).

**Step 1:** Create or open the model from the previous section and save it as a new name.

**Step 2:** Recall that only the animal **Type** column was utilized previously to make decisions. Now modify the data table **TableAnimals** by inserting a new “Integer” column from the *Standard Property* dropdown. Name this column **NumberStaff** to represent the number of vets required by the animal type. Next, add an **OBJECT LIST** column from the *Object Reference* dropdown named **ListStaff**, which will represent the list of the resources that can be selected. Set the values of the rows according to the ones in Figure 13.21. Now, the patients know which group of people can service them and the number of patients they need.

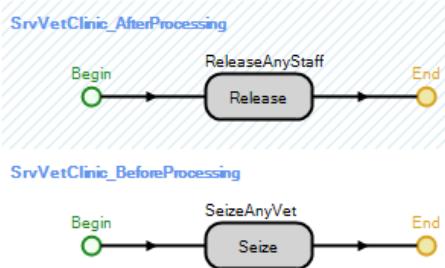
Table Animals					
	Animal	Type	Percent Type	Number Staff	List Staff
1	CatsDogs	0	47		1 ListResources
2	CowsHours	1	25		2 ListResources
3	Birds	2	20		1 ListBird
4	Iguanas	3	8		1 ListIguana

**Figure 13.21: Including the Number of Staff Needed in the TableAnimals**

**Step 3:** In the “Processes” tab, remove all the steps except one *Release* and *Seize* step in the “Before Processing” and “After Processing” add-on triggers, respectively, as seen in Figure 13.22.<sup>187</sup>

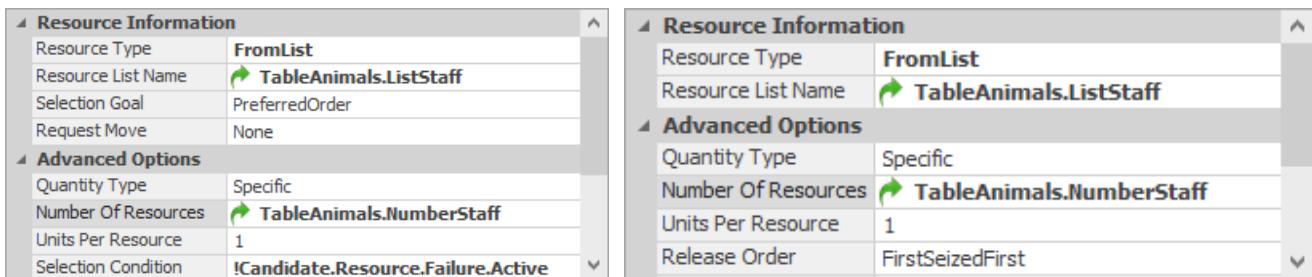
---

<sup>187</sup> Modeling the logic in this fashion has removed the two decision steps which will speed up the simulation model as well as simplify the logic when, for example, more animal types are added. Now that information is contained in the data table, eliminating the need to add additional process logic.



**Figure 13.22: Improved Process Logic for Seizing and Releasing Vet Staff Members**

**Step 4:** Modify the *Seize* and *Release* properties to utilize the referenced/associated table properties. On the *Object List Name* and *Number of Objects*, right-click and set the Referenced Property to the correct column of the table. Now, when a patient arrives to seize a staff member, it will dynamically seize the appropriate number from the correct group of resources.



**Figure 13.23: Seize and Release Properties using Referenced Properties**

**Step 5:** Save and run your model and compare the results to the previous model's results.

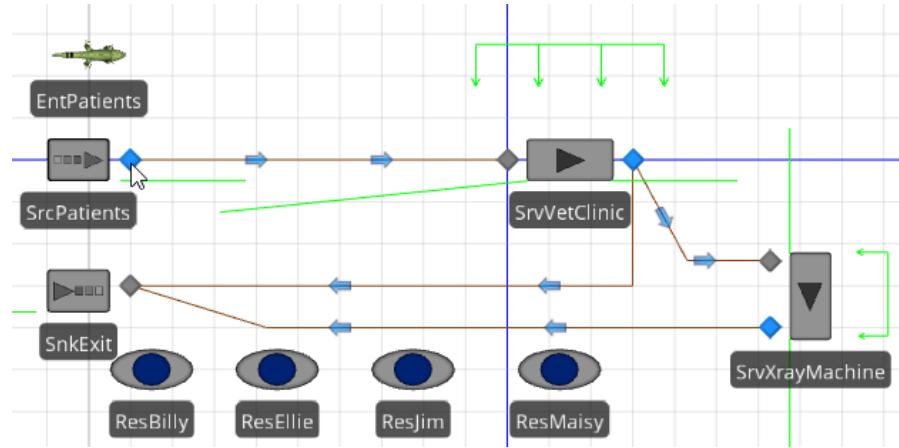
#### Part 13.4: Adding an Additional Process

Sometimes, the patients need to have an X-ray taken, which is located in a different room. For example, 25% of cats and dogs typically need X-rays, while only 15%, 0%, and 2% of cows and horses, birds, and iguanas, respectively, do. The patients will leave the care rooms and move to the X-ray machine room. The veterinarians will accompany the patients to perform the X-ray. Afterward, the veterinarians will be released to serve other patients.

**Step 1:** Add an additional SERVER named **SrvXrayMachine**, as seen in Figure 13.24.

- Set the capacity of the SERVER to one since there is only one machine
- The processing time to perform and read an X-ray follows a Pert distribution with minimum, most likely, and maximum values of 20, 25, and 35 minutes, respectively.
- Make the alignment of the processing animated queue to “Oriented Point.”
- Connect the output of the X-ray machine to **SnkExit** via a 30-meter logical PATH.
- Connect a 10-meter logical PATH from the **SrvVetClinic** output to the input of the **SrvXrayMachine**.

**Step 2:** Since each animal type has a different chance of needing an X-ray, add an additional Real column named **XrayLabPercentage** to the **TableAnimals**, as seen in Figure 13.25.

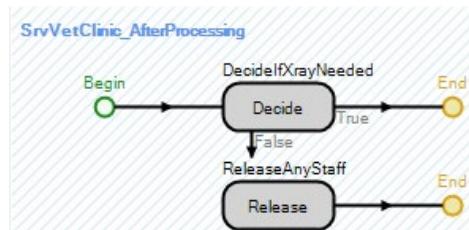


**Figure 13.24: Adding an Additional Process that uses Resources Already Seized**

Table Animals						
	Animal	Type	Percent Type	Number Staff	List Staff	Xray Lab Percentage
1	CatsDogs	0	47		1 ListResources	0.25
2	CowsHourses	1	25		2 ListResources	0.15
3	Birds	2	20		1 ListBird	0
► 4	Iguanas	3	8		1 ListIguana	0.02

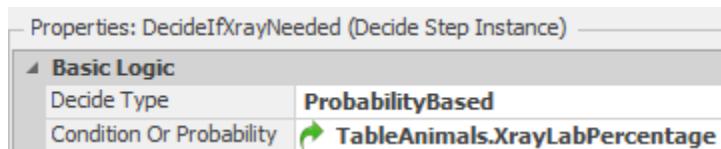
**Figure 13.25: Animal Percentages of Needing an X-ray**

**Step 3:** If the patient needs an X-ray, the staff member(s) should not be released immediately after the patient care service. The entity should move to the X-ray station SERVER, where they will be processed in the order they arrive. Once they have completed the X-ray process, the staff members are released back to the pool of available veterinarians, and the patients will exit. The first step will determine whether the patient should release the staff members (i.e., a *Decide* process step will be needed). In the “After Processing” add-on process trigger of the **SrvVetClinic**, insert a *Decide* process step. Move the *Release* step to the “False” branch (see Figure 13.26) so that the staff is not released if an X-ray machine is needed.



**Figure 13.26: Modified Processed Process Trigger**

- The *Decide* step is shown in Figure 13.27.



**Figure 13.27: The Decide Step**

**Step 4:** If the patient needs an X-ray, they release the staff members after the X-ray has finished processing them. Insert a new process for the *After Processing* add-on process trigger for the **SrvXrayMachine**. Copy and paste the *Release* step block from the **SrvVetClinic\_AfterProcessing** process to the new process.

**Step 5:** Now, patients needing X-rays must be routed to the new SERVER. By default, the patients will randomly go 50% of the time to the sink or the X-ray machine's input, owing to one's equal link weights.<sup>188</sup> Numerous functions (i.e., f(x) designation in the expression editor) are associated with the various objects. Table 13.2 gives a description of the **SeizedResources** family of functions. The base function of **SeizedResources**, which gets access to the objects that any intelligent object has seized, will invoke the underlying function **SeizedResources.NumberItems**.

**Table 13.2: Description of all the **SeizedResources** Functions**

Function	Description
<b>SeizedResources.NumberItems</b>	This function will return the current number of objects (e.g., RESOURCES, SERVERS, LINK capacities, etc.) seized and owned.
<b>SeizedResources.FirstItem</b>	All object instances in a SIMIO model have a unique numeric identifier (ID) (e.g., NODES, SERVERS, RESOURCES, etc.). This function will return the reference of the first object in the list of objects currently seized and owned by this object.
<b>SeizedResources.LastItem</b>	This function will return the reference of the last object in the list of objects currently seized and owned by this object.
<b>SeizedResources.ItemAtIndex(index)</b>	This function will return the object's reference in the list of objects currently seized and owned by this object at the specified index.
<b>SeizedResources.IndexOfItem(resource)</b>	This function will return the index of a specified resource in the list of resources currently seized and owned by this object. It will return zero if the resource has not been seized.
<b>SeizedResources.CapacityOwnedOf(resource)</b>	This function will allow you to determine the number of units of capacity for a specified resource that is currently owned by this object (i.e., it would return two if they had seized two hammers).
<b>SeizedResources.Contains(resource)</b>	This function returns True(1) if the list of resources currently seized and owned by this object includes the specified resource. Otherwise, the value False(0) is returned.
<b>SeizedResources.RequestedDestinationNode(resource)</b>	This function returns a reference to the requested destination node for a specified resource in the list of resources currently seized by this object.
<b>SeizedResources.AgregateEfficiency(type)</b>	Calculates and returns an aggregate efficiency value for the list of resources currently seized by the object. The aggregate type is an integer argument. Possible values: 0 = None, 1 = Average, 2 = Count, 3 = Maximum, 4 = Minimum, 5 = Sum.

---

<sup>188</sup> Note, we could have specified the link weights to be 1-TableAnimals.XrayLabPercentage for the link to the sink and TableAnimals.XrayLabPercentage for the link to the X-ray machine and the patients would have been routed correctly. However, this logic would only work if the patients did not need the same staff members to accompany them to the X-ray machine (i.e., they could be released before exiting the **SrvVetClinic**).

- Specify the link weight of the path leaving the **SrvVetClinic** and going to the exit (**SnkExit**) to be `ModelEntity.SeizedResources.NumberItems == 0`, which means the patient has released the staff members and can exit the system.
- Specify the link weight of the path leaving the **SrvVetClinic** and going to the X-ray machine (**SrvXrayMachine**) to be `ModelEntity.SeizedResources.NumberItems > 0`, which means the patient has still seized staff members because the patient needs to have an X-ray done.
- Note that the link weights will evaluate to one or zero thus forcing the patients to take the path only if the link weight was a one.

**Step 6:** Save and run the model and observe what happens.

### Part 13.5: Changing Processing Based on Animal Type and Vet Servicing

At this point, all of the animal types take the same amount of time to be seen by one of the vets (i.e., processing time is sampled from the same distribution). However, depending on which animal type and vet is seeing the animal, the time to be seen can vary. This is a common situation where the time depends on the resource seized (i.e., doctor, nurse, etc.). Recall that relational data tables were used previously to specify a different processing time for each part type for each of the operations in the sequence. This method works well for distinguishing processing times among different part types (i.e., animal types). Therefore, a child table containing the processing times will be created to allow the processing time to become also dependent on the resource being employed.

**Step 7:** First, select the **Type** column in the **TableAnimals** and make it the table's primary key, as seen in Figure 13.28. This table will become the parent table, and the child table will inherit the values.

Table Animals						
	Animal	Type	Percent Type	Number Staff	List Staff	Xray Lab Percentage
1	CatsDogs	0	47		1 ListResources	0.25
2	CowsHours	1	25		2 ListResources	0.15
3	Birds	2	20		1 ListBird	0
4	Iguanas	3	8		1 ListIguana	0.02

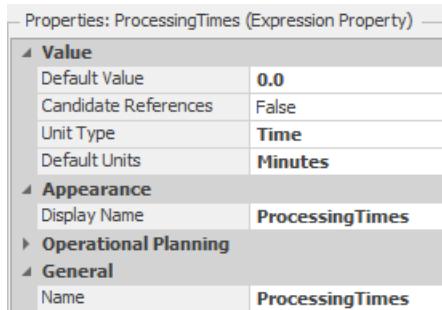
**Figure 13.28: Specify the TableAnimals Type Column as a Primary Key.**

**Step 8:** Next, add a new data table named **TableProcessingTimes** via the *Data→Tables→Add Data Table* button as seen in Figure 13.31.

- Add a new Standard *Expression Property* column named **ProcessingTimes**, as seen in Figure 13.29 making sure to set the properties *Unit Type* to “Time” and *Default Units* to “Minutes.”<sup>189</sup>

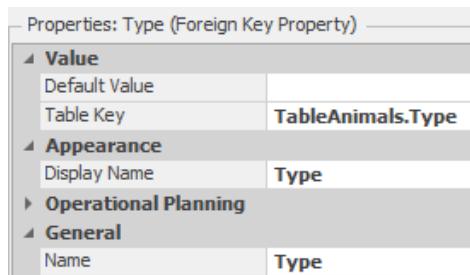
---

<sup>189</sup> It has to be an Expression property to allow distributions to be specified.



**Figure 13.29: Specifying the Processing Time Column**

- Next, add a new *Foreign Key* column named **Type** with the *Table Key* property set to `TableAnimals.Type` as seen in Figure 13.30.



**Figure 13.30: Specifying the Foreign Key to inherit Column from Parent Table**

- At first we will utilize constant times for each of the animal services. Set the time to 30, 60, 20, and 40 minutes for dogs, cows, birds and iguanas respectively to correspond to Figure 13.31.

Table Animals		Table Processing Times	
	ProcessingTimes (Minutes)		Type
1	30		0
2	60		1
3	20		2
4	40		3

**Figure 13.31: Processing Time Table for the Different Animal Types**

**Step 9:** Next, we need to utilize the new processing time information in the model. Unlike prior cases where we were using the next sequence of the TRANSFERNODE to set the particular row in the child table, we will need to specify it directly (i.e., set the *Processing Time* property for row one by `TableProcessingTimes[1].ProcessingTime` of **SrvVetClinic**). Without the relational table, the expression would always be set to the first entry in the whole table. Since this is a related table, it will set it to the first entry of the inherited rows which currently only has one row as seen in Figure 13.33 which shows for child table for an entity which has been assigned a bird.<sup>190</sup>

<sup>190</sup> Notice, an error would occur if we just specify `TableProcessingTimes.ProcessingTime` as was done in a previous chapter without first setting the row via a *Set Row* step or a *By Sequence* from the TRANSFERNODE.

Process Type	Specific Time
► Processing Time	TableProcessingTimes[1].ProcessingTimes

Figure 13.32: Setting the Processing Time to be the First Row of the Child Table

CowsHours...	1	25
Birds	2	20
Table Processing Times		
1	ProcessingTimes (Minutes)	Type
*	20	key

Figure 13.33: Showing the Child Table Associated with the Bird Row

**Step 10:** At this point, we could run the model, but it may be difficult to determine if the processing time is being assigned correctly.<sup>191</sup> Therefore, we will set the priority<sup>192</sup> state variable of each entity to reflect their processing time. In the *SrcVetClinic\_Processing* process, insert an *Assign* step after the *Seize* step to set the patient's PRIORITY state variable, as seen in Figure 13.34.<sup>193</sup>

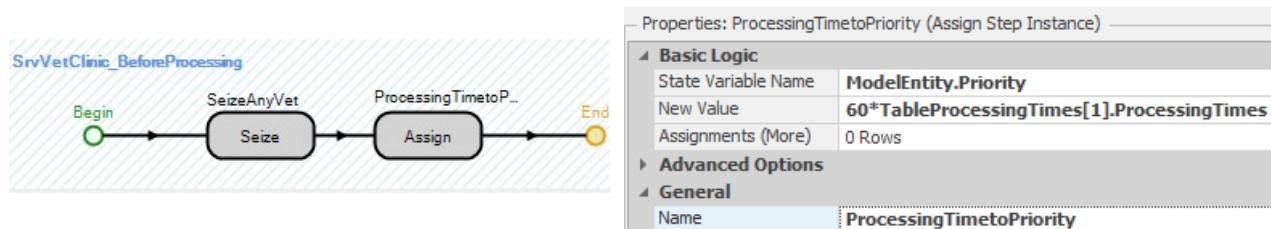


Figure 13.34: Specifying the Processing Time for Display Purposes

**Step 11:** In order to see the set priority value, we will add a status label to the animal that displays the processing time for each animal.<sup>194</sup> Select the MODELENTITY **EntPatients** and then insert a new *Animation*→*Status Label*, which associates/attaches the label to the entity. Specify the expression of the label as the entity's priority state variable.

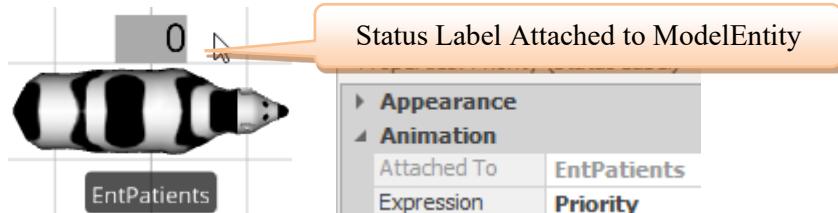


Figure 13.35: Inserting a Status Label Attached to the Entity

*Question 8:* Why do we have to multiply the processing time by 60?

**Step 12:** Save and run the model, observing the status label times.

<sup>191</sup> We could turn on the trace and watch what the processing time delay was when the animal went into service.

<sup>192</sup> Recall that “priority” is a SIMIO-defined state variable for entities.

<sup>193</sup> We are multiplying by 60 because SIMIO has converted all times to hours internally.

<sup>194</sup> SIMIO now provides a *Dynamic Label Text* property under the *Animation* section which can be used to a similar thing but the label appears below the MODELENTITY.

*Question 9:* Are the correct times being assigned to the different animals?

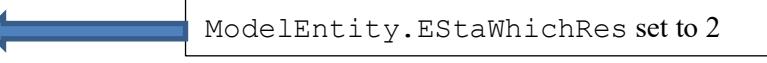
---

**Step 13:** Now, the model can handle the processing time based on the animal type. In many situations, the processing time may also depend on the resource (i.e., vet) you have seized. The notion of position within the list of resources will be used to determine the processing time. First, select the MODELENTITY in the [Navigation] panel and add a new DISCRETE INTEGER STATE variable from the Definitions→State section named EStaWhichRes. This state variable will be assigned the position of the resource that has been seized in the particular resource list.

For example, if a current patient who is a dog has seized Ellie from the **ListResource** list the patient's **EStaWhichRes** variable would be assigned a value of "2" to correspond that **ResEllie** is in the second position of this list as seen in Figure 13.36.

Position	ListResource
1	ResBilly
2	ResEllie
3	ResJim
4	ResMaisy

ModelEntity.EStaWhichRes set to 2



**Figure 13.36: Illustrating how Resource Position will be used**

**Step 14:** Currently, the data table **TableProcessingTimes** contains only time for each animal type. Modify the table to include a different processing time for each vet (i.e., resource) and each animal type, as seen in Figure 13.37.<sup>195</sup> Since four different resources are capable of processing dogs, there are four different processing times (i.e., 31 minutes for Billy, 32 minutes for Ellie, 33 minutes for Jim, and 34 minutes for Maisy) for animal type "0" (i.e., cats and dogs). For birds and iguanas, only two vets can see each of those types of patients. Therefore, only two different processing times are specified for types "2" and "3". As Figure 13.37 illustrates, Jim will process Iguanas in 42 minutes while Ellie takes 43 minutes.

---

<sup>195</sup> One of the easiest ways to create the table is to copy it from an Excel worksheet where the table can be created easily. Tables can be copied to and from Excel. To copy from Excel, first copy the two dimensional table in Excel and then highlight all the current values in the SIMIO data table before pasting the new values which will overwrite any existing one and then create the any new rows as needed.

Table Animals		Table Processing Times	
		ProcessingTimes (Minutes)	Type
	1	31 ← Billy	🔑 0
	2	32 ← Ellie	🔑 0
	3	33 ← Jim	🔑 0
	4	34 ← Maisy	🔑 0
	5		🔑 1
	6		🔑 1
	7		🔑 1
	8		🔑 1
	9		🔑 2
	10		🔑 2
	11		🔑 3
	12		🔑 3

**Figure 13.37: Add more Processing Times for Each Resource**

Figure 13.38 shows the child-related table for animal type zero (i.e., cats and dogs) when expanding out the type “0” from the parent table **TableAnimals**. As you can see, only four processing times are related to type zero. Now, going back to the example in Figure 13.36, Ellie would correspond to the second position in the list, which translates to row two of the related table (i.e., 32 minutes).

Table Animals		Table Processing Times		
	Animal	🔑 Type	Percent Type	Num
▶ 1	CatsDogs	0	47	
Table Processing Times				
	ProcessingTimes (Minutes)	Type	TableProcessingTimes [ModelEntity.EStaWhichRes]. ProcessingTime	
▶ 1	31	🔑 0	Ellie	
2	32	🔑 0		
3	33	🔑 0		
4	34	🔑 0		

**Figure 13.38: Showing the Child Sub Table for Cats and Dogs Animal Type**

**Step 15:** The only step left is appropriately assigning the **EStaWhichRes** variable based on which resource was seized. To accomplish this task, we will employ a *Search* process step where Table 13.3 explains all of the properties associated with the step.<sup>196</sup> The *Search* process step can search a collection of items (see Table 13.4) and return **TOKENS** associated with the items in the collection.

<sup>196</sup> Recall the **TOKEN** executing the process steps is typically associated with an entity. Therefore any steps (i.e., Transferring, Setting Nodes, etc.) are only applied to the **MODELENTITY** associated with the **TOKEN**. Often we may want to affect other objects (resources seized, etc.) besides the current entity, the search step can be used to create new **TOKENS** associated with other objects. In some process steps, the action can be applied to the parent object of the **TOKEN**'s associated object (e.g., the **SERVER** the logic resides).

**Table 13.3: The Properties of the Search Process Step**

Property	Description
<i>Collection Type</i>	The collection type to search (See Table 13.4 for more information).
<i>Search Type</i>	The type of search to perform (Forward, Backward, MinimizeReturnValue, and MaximizeReturnValue) <sup>197</sup>
<i>Match Condition</i>	Optional match condition that can be used to filter the items in the collection before the search begins. Don't forget to use the keyword CANDIDATE in the condition.
<i>Search Expression</i>	An expression that is evaluated for each item found by the search. The total sum for this expression is returned in the ReturnValue of the original TOKEN. One could save a particular Resource and seize the exact same one later.
<i>Starting Index</i>	One-based index to start searching. By default, it is one for Forward searches, while the number in the collection is for Backward searches.
<i>Ending Index</i>	One-based index-to-end searching to assist in narrowing the search within the collection.
<i>Limit</i>	An expression that specifies the maximum number of items that can be found and returned (e.g., Infinity or one)
<i>Save Index Found</i>	Specify an optional discrete state variable to save the last item found
<i>Save Number Found</i>	Specify an optional discrete state variable to save the number of items found.

**Table 13.4: Different Types of Collections that Can Be Searched**

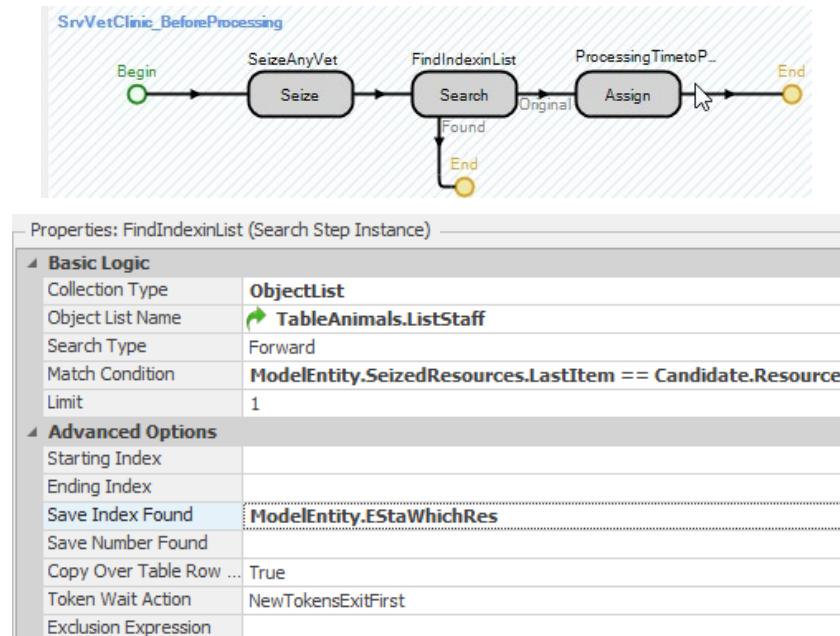
Collection Type	Description
EntityPopulation	Search a particular entity population (i.e., MODELENTITY, WORKER, or VEHICLES)
QueueState	Search the objects currently in one of the queues
TableRows	Search the rows in a Data table
ObjectList	Search the objects in a list (e.g., ResourceList, etc.)
NodeList	Search all nodes in a particular list.
TransporterList	Search all transporters in a particular list.
SeizedResources	Search the objects currently seized by the Parent Object, Associated Object, or Specific Object
ResourceOwners	Search the objects that currently own (i.e., seized) a Parent Object, an Associated Object or a Specific Object. Search all animals who have seized the <b>SrvVetClinic</b> capacity.
NetworkLinks	Search all links of a particular network.
NodeInboundLinks	Search all the inbound links of a particular node.
NodeOutboundLinks	Search all the outbound links of a particular node.
ObjectInstance	Search all objects of a particular class (e.g., all entities that are currently created).

Insert a *Search Step* between the *Seize* and the *Assign* (see Figure 13.39). The “Original” branch is associated with the original TOKEN (i.e., customer), while the “Found” branch will be invoked for a TOKEN associated with each item found.<sup>198</sup> The *collection type* will be an “object list” with the list name based on the animal type (**TableAnimals.ListStaff**). We only want to match the last resource seized by the patient with a

<sup>197</sup> *MinimizeReturnValue* and *MaximizeReturnValue* search values will search the collection and return up to the limit of items specified that minimizes or maximizes the *Return Value* expression property.

<sup>198</sup> The original token will not move forward until all the searching is done.

resource from the resource list.<sup>199</sup> If it finds a match, it will save the index (i.e., row position) in the MODELENTITY's **StaWhichRes** state variable.



**Figure 13.39: Adding a Search Step to Find the Index Position of the Seized Resource in the List**

**Step 16:** Now that the **EStaWhichRes** has been set, you need to modify the *Processing Time* property of the **SrvVetClinic** and the **Assign** step to utilize the following expression (i.e., change the row index of “1” to **ModelEntity.EStaWhichRes**).

```
TableProcessingTimes[ModelEntity.EStaWhichRes].ProcessingTimes
```

**Step 17:** Save and run the model observing the status label times to see if different processing times are being assigned based on animal type (i.e., in the 30, 60, 20, and 40-minute ranges) as well as which vet (i.e., a resource is serving the patient).

## Part 13.6: Changing the Resource Allocation Selection

In the original description, Jim was the preferred staff member to see Iguanas, while Ellie could see them if he was busy. The clinic is considering a new policy that when Jim becomes available, he will check to see if an Iguana is waiting, and if so, he will service these patients first before seeing the next patient. A “first-in and first-out” ranking scheme is currently being used by the **SrvVetClinic** to service the patients.

RESOURCES like SERVERS have a static ranking rule<sup>200</sup> that orders the requests waiting to be allocated capacity from this object. The SERVER and RESOURCE will process them in the order in which the entities are sorted.<sup>201</sup>

<sup>199</sup> Note the use of the CANDIDATE keyword to represent each member of the collection being searched (i.e., like a wildcard \*). Also, in the case of a cow or horse which requires two vets, the second vet seized will be lower in the list and our assumption would reflect less seniority and therefore slower.

<sup>200</sup> The static ranking rules are First-in First-out (Default), Last-in Last-Out, Smallest Value, and Largest Value, where an expression provides the value for the last two rules (e.g., **ModelEntity.Priority**)

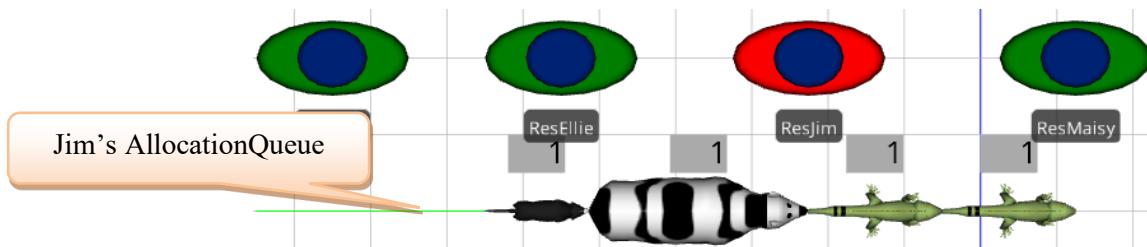
<sup>201</sup> Also there is a dynamic selection rule, which will be considered in a later chapter.

**Step 1:** Change the static ranking rule of the RESOURCE **ResJim** to be the “largest value first.” The ranking expression should be the referenced property animal type since iguanas will always be serviced first if they have requested service.<sup>202</sup> The static ranking rule works because the type of the animal does not change while the patient is waiting for a vet (see Figure 13.40).

Properties: ResJim (Resource)	
Resource Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	Largest Value First
Ranking Expression	TableAnimals.Type
Dynamic Selection Rule	None

**Figure 13.40: The Largest Value Static Ranking Rule**

**Step 2:** To visualize the ranking of **ResJim**, select **ResJim** and insert an animated queue (i.e., ALLOCATIONQUEUE) from the *Contents* section of the *Attached Animation* tab. The allocation queue is the queue that contains the requests for allocation of capacity of an object (e.g., RESOURCE, SERVER, WORKER, VEHICLE, COMBINER, etc.). Make the queue long enough to accommodate the size of the animals, as seen in Figure 13.41, which is drawn from right to left.



**Figure 13.41: Allocation Queue of ResJim**

**Step 3:** Save and run the model and observe what happens. The percentage of time Iguanas arrive may need to be altered to verify the model works the intended way.

*Question 10:* Are there any issues with ranking the queues this way?

---

*Question 11:* Did you observe cows jumping ahead of dogs in the queue?

---

**Step 4:** Jim should service the Iguanas first and then service the remaining two animals in order of their arrival. One way to accomplish this ranking is to modify the *Ranking Expression* property to the value in Figure 13.42. The logical expression will give all iguana patients a value of one, while all other patient types would be zero, and therefore, preserve the arrival order to break the ties among remaining animals (arrival order is the tie-breaking for static ranking rule).

<sup>202</sup> If **ResJim** is busy processing a patient and an iguana is at the front of the server’s queue then **ResEllie** may still see them, which is acceptable. The goal is to maximize **ResJim’s** chances of seeing Iguana patients but not to eliminate her’s completely. If only **ResJim** could see iguanas then the **ListIguana** could be changed to only have Jim as a resource.

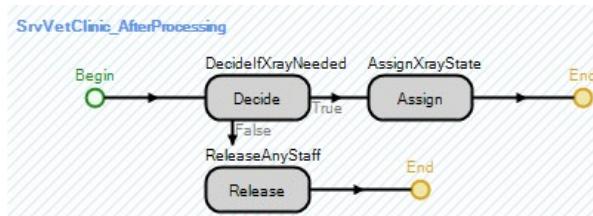
Ranking Rule	<b>Largest Value First</b>
Ranking Expression	TableAnimals.Type==3

**Figure 13.42: Modifying the Ranking Rule to not Service Cows before Dogs**

**Step 5:** Save and run the model and observe what happens now.

### Part 13.7: Commentary

- A common modeling technique is adding additional state variables to the general entity to hold values or states. In the second part, an *EStaXray* discrete state variable could have been added to the entity. In the True branch of the “*After Processing*” add-on trigger, as seen in Figure 13.43, an *Assign* step could have been added to assign the value of one to the new *EStaXray* state variable to show that an X-ray is needed.



**Figure 13.43: Using a State Variable to Assign Whether an X-ray is Needed or Not**

- The link weights could have been specified as  $(1-\text{ModelEntity.EStaXray})$  and  $(\text{ModelEntity.EStaXray})$  for the links to the exit *SINK* and the X-ray *SERVER* machine, respectively. Again, the link weights will take on zero or one and will never be the same. If the patient does need an X-ray, then  $(1-\text{ModelEntity.EStaXray})$  would evaluate to one while  $(\text{ModelEntity.EStaXray})$  would evaluate to zero.
- To some extent, one could have used the *Secondary Resources* to seize and release the vets. One could have easily used *Before Processing* under *Other Resource Seizes* and *After Processing* under *Other Resource Releases* with the same input as in Figure 13.23. However, the ability to release or not release if the animal needed a vet would require a little more modeling if you were trying to avoid processes altogether. It is easy not to release the Vets at the clinic and then only release them under the *After Processing* under *Other Resource Releases* of the **SrvXrayMachine** if the animal needs an X-ray. However, there is no mechanism to release the vets for those who do not. You would have to add an additional *SERVER* between the **SrvVetClinic** and the **SnkExit** that would release them. As has been seen in other chapters, one has more control and flexibility in using processes.

# Chapter 14

## A Mobile Resource: The Worker

---

The previous chapter illustrated the flexibility of RESOURCES and RESOURCE LISTS to model very complicated relationships. However, these types of resources were fixed because they did not move dynamically throughout the system model. When the patients needed an X-ray, the vets were not released, and the patient moved to the X-ray machine room, but the animation did not show the resources moving along with the patient. Also, in the DMV problem, resources were used in multiple locations, but the travel time between the two locations was not considered. SIMIO has the ability to have multiple types of objects (i.e., entities, workers, and vehicles) that can act as resources (i.e., objects that can be seized and released) and move throughout the system, similar to an entity.

### Part 14.1: Routing Patients

Again, we return to the veterinary clinic. However, this time, the clinic will behave more consistently with a doctor's office where patients arrive, check-in at the front desk, and then wait in the waiting room for an available room. The patients will move to one of four examination rooms once one becomes available. Then, one veterinarian will start from their office, dynamically move to the rooms as needed, and service the patients as requested. The veterinarian will be modeled as a movable resource that needs to be seized and released.

**Step 1:** Create a new model that contains one SOURCE (**SrcPatients**), a SINK (**SnkExit**), and five SERVERS (**SrvVetWaiting**, **SrvRoom1**, **SrvRoom2**, **SrvRoom3**, and **SrvRoom4**).

**Step 2:** Insert a new MODELENTITY named **EntPatients** into the model. The patients travel at a constant rate of 4.5 km per hour or 4500 meters per hour.

**Step 3:** Insert the SERVER rooms into a two-hall configuration, with the waiting room serving the two hallways, as seen in Figure 14.1.<sup>203</sup>

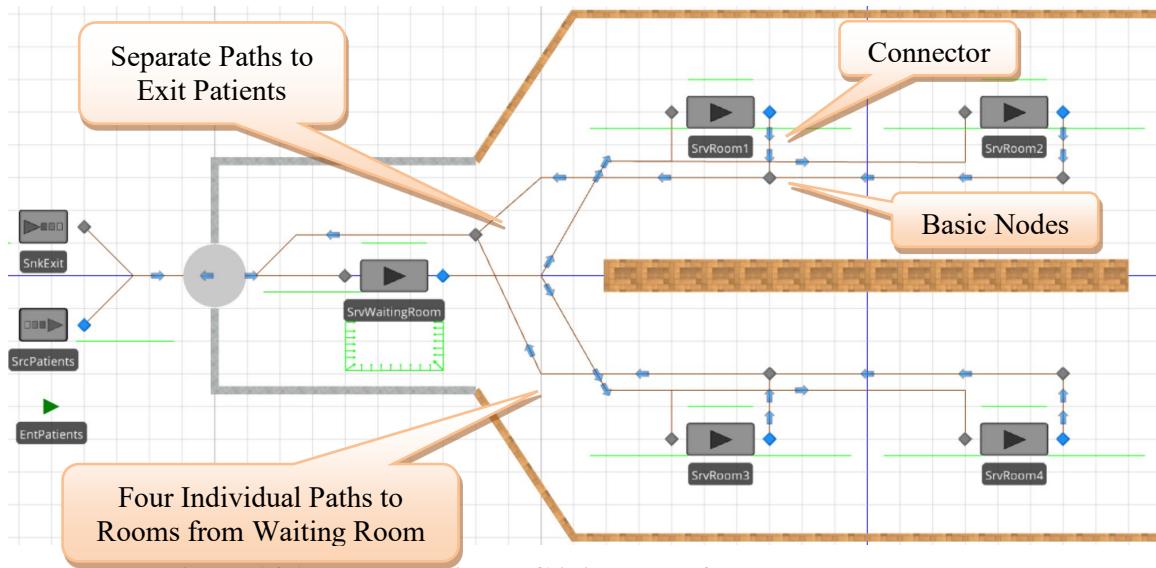
- The capacities of all the servers should be one.
- The check-in process at the **SrvVetWaiting** uniformly takes between one and four minutes before they are placed into the waiting room.
- Each patient takes between 10 and 20 minutes to be seen, most of which take 12 minutes.
- It has been observed that patients arrive exponentially with an interarrival time of 6.5 minutes.

**Step 4:** Insert a wall in the middle of the rooms to add a more visual cue for the hallway SERVERS separation, as seen in Figure 14.1.<sup>204</sup> Make sure to specify a height of 3.5 meters and a width of one meter and give the wall a texture. You will notice we inserted a different line (i.e., window texture) around the waiting room, plus added a 3-D door as the entrance.

---

<sup>203</sup> To be more efficient, create one room named **SrvRoom** with all the correct properties and then copy it three times, changing the name of the original one to **SrvRoom4** accordingly.

<sup>204</sup> From the *Drawing* tab, insert a Polyline to represent the walls. You should do separate lines for the outside walls so you can add different textures. Change the width of the outside wall lines under the *Drawing* tab to 0.25 meters and a height of 3.5 meters.



**Figure 14.1: New Veterinary Clinic Model for Moveable Resources**

**Step 5:** Connect the output of the **SrvVetWaiting** to each of the inputs to the four rooms using four PATHS named **PathRoom1**, **PathRoom2**, **PathRoom3**, and **PathRoom4**, respectively.

- The distance between the waiting room and patient rooms one and three is 15 meters; it is 30 meters for the other rooms.
- Select all four PATHS and set the traveler capacity to one and the speed to 4.5 kilometers per hour.

Properties: PathRoom1,PathRoom2,PathRoom3,PathRoom4	
<b>Travel Logic</b>	
Type	Unidirectional
Initial Traveler Capacity	<b>1</b>
Entry Ranking Rule	First In First Out
Drawn To Scale	<b>False</b>
> Logical Length	
Allow Passing	True
Speed Limit	<b>4.5</b>
Units	Kilometers per Hour

**Figure 14.2: Setting up the Patient Entrance Paths**

**Step 6:** Next, provide a means for the patients to leave the examining rooms and exit the clinic using paths and the same distances from the waiting room to the rooms for the rooms to the exit. BASICNODES can be used to assist in directing the patients out, as seen in Figure 14.1.<sup>205</sup> We can use common exit paths (i.e., paths between rooms one and two and four and three) for the output since we do not need to constrain the paths to only have one person traveling on them, as was the case with the input paths.

**Step 7:** Connect the SOURCE (**SrcPatients**) via a 10-meter PATH to the **SrvVetWaiting** and use a 10-meter PATH to connect the last BASICNODE to the **SrvExit**.

**Step 8:** Modify the OUTPUTBUFFER.CONTENTs queue of the **SrvVetWaiting** server to have an *Oriented Point* alignment with several locations with the *Keep in Place* option checked which will improve the animation.

<sup>205</sup> CONNECTORS are used to connect the outputs of the rooms to the BASICNODES while paths with appropriate logical lengths of 15 meters are used to connect the basicnodes from rooms two to one, three to four, one to the waiting room, and three to the waiting room.

**Step 9:** As before, patients should select a room that is currently available. Create a new node list via the *Definitions* tab named **ListRooms**, allowing patients to choose an available room. Figure 14.3 shows the list, noting the order of preference is one, three, two, and then four (i.e., fill up the closest rooms first if more than one room is available).

Nodes	
	ListRooms Nodes
Node	
Input@SrvRoom1	
Input@SrvRoom2	
Input@SrvRoom3	
Input@SrvRoom4	

**Figure 14.3: Creating a Node List of Examination Rooms**

**Step 10:** For the output **TRANSFERNODE** of the waiting room (**Output@SrvVetWaiting**), specify the *Entity Destination Type* to select a node “*From a list*” using the **ListRooms** as the *Node List Name* property. The goal is to route patients to an available room. Therefore, specify the *Selection Goal* as the “Smallest Value.” In previous chapters, we used an expression to determine the total number of entities routing to the input node, waiting in the input buffer, plus those currently being processed by the **SERVER**.

**Step 11:** However, a series of **AssociatedStation**<sup>206</sup> functions (see Table 14.1) can be used as part of the dynamic routing selection for all fixed objects that have external input nodes (e.g., **SERVERS**). These external input nodes are associated with an internal **STATION** (e.g., **Input@SrvRoom1** is associated with the input buffer station of **SrvRoom1**).

**Table 14.1: Explanation of the AssociatedStation Functions**

Function	Description
<b>AssociatedStation.Capacity</b>	This function returns the location’s capacity inside the associated object (e.g., the input buffer capacity of the server). <sup>207</sup>
<b>AssociatedStation.Capacity.Remaining</b>	This function returns the current available unused capacity of the server’s input buffer.
<b>AssociatedStation.Contents</b>	This function will return the current number of entities in the location (e.g., the number in the input buffer of a server ( <b>InputBuffer.Contents</b> )). <sup>208</sup>
<b>AssociatedStation.EntryQueue.NumberWaiting</b>	This function will return the current number of entities waiting to enter the location ( <b>InputBuffer.EntryQueue</b> ).
<b>AssociatedStationLoad</b>	This function will return the “load,” which is the sum of entities on their way to the node, waiting to enter and being processed (i.e., input buffer and processing stations).
<b>AssociatedStationOverload</b>	This function returns how much a location is overloaded, which

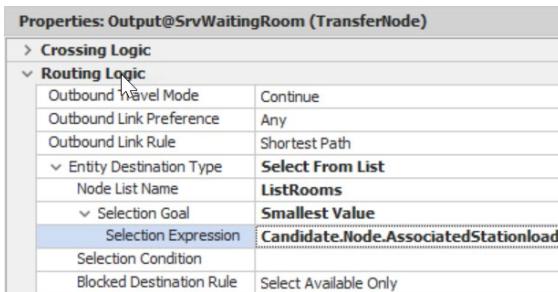
<sup>206</sup> When you access the **AssociatedStation** functions for an external node, you are given access to the actual station and all of its properties, states, and functions.

<sup>207</sup> The capacity of the **SERVER** (i.e., the **Processing STATION**) is not included unless the **input buffer capacity is zero**. Then, all the functions report values associated with the **Processing STATION** instead of the **InputBuffer STATION**. Note that an infinite capacity is represented by the maximum integer value of 2147483647.

<sup>208</sup> The **AssociatedStation.Contents** for the input buffer does not include the entities that are currently being processed by the **SERVER** or in the **OUTPUTBUFFER** station. The **AssociatedStationLoad** does not include those in the **OUTPUTBUFFER STATION**.

	equals the “Load” minus “Capacity,” which can be both positive (overloaded) or negative (underloaded).
--	--

Use the expression (`Candidate.Node.AssociatedStationload`), which returns the total number of entities heading to the room, waiting to enter the room, waiting at the room plus the current number of entities being processed by the room as the *Selection Expression* property as seen in Figure 14.4.



**Figure 14.4: Specifying the *Selection Expression* to Choose the Room with the Smallest Number of Patients**

**Step 12:** Save and run the model for 24 hours to observe what happens.

*Question 1:* Does the model behave the way it was intended?

---

**Step 13:** Patients seem to wait by the examining rooms after being checked in instead of in the waiting room, which was different than what was intended. Therefore, we need to force the patients to wait in the waiting room until a room is available in a similar fashion as was done in Chapter 13. Select each of the rooms and change the *Input Buffer*<sup>209</sup> capacity to “0”. Select the output node of the **SrvVetWaiting** (i.e., **Output@SrvVetWaiting**) and make sure the *Blocked Destination Rule* is set to “Select Available Only.”

**Step 14:** Run the model to observe what happens. You might need to increase the arrival rate to see if there are any issues (i.e., set the interarrival rate to 0.5) and reduce the processing time in the waiting room.

*Question 2:* Does the model behave the way it was intended, or are there animation issues still present?

---

The patients will now wait properly in the waiting room since the input buffer capacity is zero, which prevents patients from heading and waiting outside the room doors.

## Part 14.2: Using a Worker as a Resource

The current model addresses the exam rooms but does not model the veterinarians. A fixed RESOURCE object similar to previous chapters could be used to constrain the processing to only one staff member, but the object is stationary. Since the veterinarian needs to travel between rooms, a dynamic moving resource would better reflect the real system. WORKERS (and VEHICLES) in the system can be specified as resources (i.e., they can be seized and released like fixed RESOURCES), but they can also travel throughout the network to service.

**Step 1:** Insert a new TRANSERNODE named **TOffice** to represent the vet’s office at the end of the wall near rooms two and four.

---

<sup>209</sup> You can select all four rooms and then set the *Input Buffer* to “0” under the *Buffer Capacities* properties at once. Here the spreadsheet view is also useful.

**Step 2:** From the [Project Library], insert a new WORKER named **WkrVet**. Change the symbol of the WORKER to an appropriate figure if you would like. Modify the properties as specified in Figure 14.5.

<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2">Properties: WkrVet (Worker)</th></tr> </thead> <tbody> <tr> <td colspan="2"> <b>Resource Logic</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Capacity Type</td> <td>Fixed</td></tr> <tr> <td>Ranking Rule</td> <td>First In First Out</td></tr> <tr> <td>Dynamic Selection Rule</td> <td>None</td></tr> <tr> <td>Park While Busy</td> <td><b>True</b></td></tr> </table></td></tr></tbody> </table>	Properties: WkrVet (Worker)		<b>Resource Logic</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Capacity Type</td> <td>Fixed</td></tr> <tr> <td>Ranking Rule</td> <td>First In First Out</td></tr> <tr> <td>Dynamic Selection Rule</td> <td>None</td></tr> <tr> <td>Park While Busy</td> <td><b>True</b></td></tr> </table>		Capacity Type	Fixed	Ranking Rule	First In First Out	Dynamic Selection Rule	None	Park While Busy	<b>True</b>	
Properties: WkrVet (Worker)													
<b>Resource Logic</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Capacity Type</td> <td>Fixed</td></tr> <tr> <td>Ranking Rule</td> <td>First In First Out</td></tr> <tr> <td>Dynamic Selection Rule</td> <td>None</td></tr> <tr> <td>Park While Busy</td> <td><b>True</b></td></tr> </table>		Capacity Type	Fixed	Ranking Rule	First In First Out	Dynamic Selection Rule	None	Park While Busy	<b>True</b>				
Capacity Type	Fixed												
Ranking Rule	First In First Out												
Dynamic Selection Rule	None												
Park While Busy	<b>True</b>												
<b>Travel Logic</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Initial Desired Speed</td> <td><b>4.5</b></td></tr> <tr> <td>Units</td> <td><b>Kilometers per Hour</b></td></tr> <tr> <td>Initial Travel Mode</td> <td>Network If Possible</td></tr> <tr> <td>Initial Network</td> <td>Global</td></tr> <tr> <td>Network Turnaround Method</td> <td>Exit &amp; Re-enter</td></tr> <tr> <td>Free Space Steering Behavior</td> <td>Direct To Destination</td></tr> </table>		Initial Desired Speed	<b>4.5</b>	Units	<b>Kilometers per Hour</b>	Initial Travel Mode	Network If Possible	Initial Network	Global	Network Turnaround Method	Exit & Re-enter	Free Space Steering Behavior	Direct To Destination
Initial Desired Speed	<b>4.5</b>												
Units	<b>Kilometers per Hour</b>												
Initial Travel Mode	Network If Possible												
Initial Network	Global												
Network Turnaround Method	Exit & Re-enter												
Free Space Steering Behavior	Direct To Destination												
<b>Routing Logic</b> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Initial Priority</td> <td>1.0</td></tr> <tr> <td>Initial Node (Home)</td> <td><b>TOffice</b></td></tr> <tr> <td>Idle Action</td> <td><b>Park At Home</b></td></tr> <tr> <td>Off Shift Action</td> <td>Park At Node</td></tr> </table>		Initial Priority	1.0	Initial Node (Home)	<b>TOffice</b>	Idle Action	<b>Park At Home</b>	Off Shift Action	Park At Node				
Initial Priority	1.0												
Initial Node (Home)	<b>TOffice</b>												
Idle Action	<b>Park At Home</b>												
Off Shift Action	Park At Node												

 Change the *Park While Busy* property to **True**, which will force the worker to park when it is busy servicing a customer. This will help minimize blocking issues on paths.  Change the *Initial Desired Speed* to 4.5 kilometers per hour.  Specify that the *Initial Node (Home)* will be the office (i.e., **TOffice**) after performing Step 2 below.  Specify the *Idle Action* property as “**Park At Home**.” |

**Figure 14.5: Specifying the Worker Properties**

**Step 3:** If you run the model now, the **WkrVet** will stay in its office but not actively seek any patients. The patients will need to request the **WkrVet** in order to be served. When the patient goes into processing, they need to seize the **WkrVet** and then release it once they have finished processing.<sup>210</sup>

- From the *Processes* tab, create a new process named **SiezeVet** and insert a *Seize* step that will request a **WkrVet** to service a patient, as seen in Figure 14.6. Select all four room servers and select the new process **SiezeVet** as the *Before Processing* Add-on process trigger.



**Figure 14.6: Seizing the Worker**

- Create a new Process called **ReleaseVet**. It will be triggered in the Add-on process trigger “*After Processing*.” Insert a *Release* step that will release the Vet to service a new patient, as seen in Figure 14.7.



**Figure 14.7: Releasing the Worker**

- In the dropdown list for the “*After Processing*” Add-on process trigger for all four rooms, just select the **ReleaseVet** process to release the veterinarian after the processing has finished.

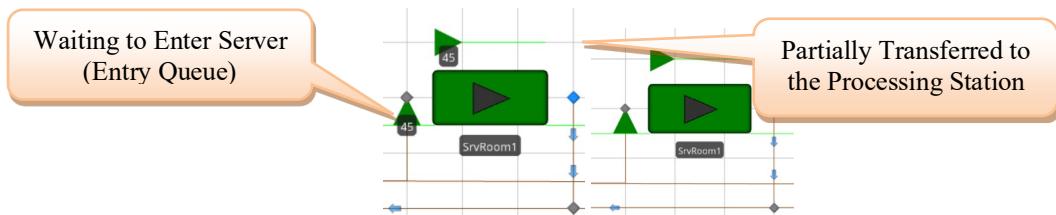
**Step 4:** Save and run the model and observe the veterinarian.

<sup>210</sup> Again we will utilize the option that gives you the most control (i.e., processes). However, in this case *Secondary Resources* could be used and is demonstrated in the commentary.

*Question 3:* Does the vet get seized and released correctly, and does the vet move to the appropriate room?

*Question 4:* Did you notice any animation issues with the patients in the room?

Figure 14.8 shows an animation issue of patient 45 waiting for a vet potentially in two places, which did not occur in the previous models. In those models, the SERVERS were the only constraining resource, but suppose we now need a room (i.e., the SERVER) and a vet (a RESOURCE). When we try to seize the vet (a resource) in the *Before Processing* add-on process trigger, and the input buffer is zero, the entity can become stuck in a limbo state if the room (i.e., SERVER) is available but the vet is not immediately available. Thus, the entity cannot enter the input buffer because its capacity is zero, and the vet is unavailable yet has seized the Server's capacity. The entity has not been completely transferred into the processing station. An entity waiting for a station capacity (in this case, the input buffer) to become available will wait in an "Entry Queue." Entry queues are used whenever an entity cannot enter a station because there is no capacity available. The animation anomaly occurs because the entity is waiting in the entry queue and has not completely transferred into the processing station owing to the need for secondary resources.



**Figure 14.8: Animation Anomalies**

**Step 5:** We could change the seizing of the vet to occur in the "*Processing*" add-on process trigger, which happens after the entity has been completely transferred into the processing station. If you want to try it, select **SrvRoom1**, remove the *Before Processing*<sup>211</sup> Add-on process trigger, and then select the same process from the dropdown of the *Processing* add-on process trigger. Run the model and observe what happens in the room.

**Step 6:** The animation anomaly has been removed, but the patient now waits in the processing station for the vet, which is processed when the vet arrives. However, this can be confusing and cause issues with statistics since both waiting and processing are combined now. Table 14.2 describes the options for choosing a MODELENTITY's destination node from a list. The *Selection Condition* and *Blocked Destination Rule* properties can be used to control which nodes are available for possible selection as the destination node.

<sup>211</sup> You can remove an add-on process trigger either by using the right click reset menu option or selecting the trigger name and then deleting it.

**Table 14.2: Choosing Entity's Destination via Select from List**

Routing Logic	Description
<i>Selection Goal</i>	The goal determines the logic to utilize when selecting a destination node for an Entity. This can be random or cyclic (i.e., select the next node in the list), preferred (i.e., always choose the first nodes in the list unless a <i>Selection Condition</i> eliminates a node), smallest and largest distance, and smallest and largest values.
<i>Selection Expression</i>	An expression is evaluated for each candidate node destination when the <i>Selection Goal</i> property is set to the “Smallest or Largest Value.”
<i>Selection Condition</i>	An optional condition that is evaluated for each candidate destination node. This expression must evaluate to true for the node to be considered under the <i>Selection Goal</i> .
<i>Blocked Destination Rule</i>	Determines what happens if the destination nodes are blocked. The default value, “ <b>Select Available Only</b> ,” will only evaluate nodes to be selected that are currently not blocked, while “ <b>Select Any</b> ” will select any node regardless of whether it is blocked or not based on the <i>Selection Goal</i> . The “ <b>Preferred Available</b> ” will first select among all non-blocked nodes and, if all the nodes are blocked, will revert back to select any. A node is considered blocked if the node is an input node of an object and the number of entities assigned to the station is greater than or equal to the capacity of the object (i.e., <code>AssociatedStationLoad ≥ AssociatedStation.Capacity</code> ).

Table 14.3 looks at all possible scenarios associated with the server’s input buffer size and the values of the *Selection Condition* and *Blocked Destination Rule* of the TRANSERNODE. The “Modeling Result” column represents what happens to the additional entities when the rooms all currently have one patient. The *Selection Condition* will eliminate all nodes (i.e., filter the list) that do not meet the criteria before the selection goal is used, while the *Blocked Destination Rule* may not allow the node from being selected if the node is currently blocked even though it was not filtered out.

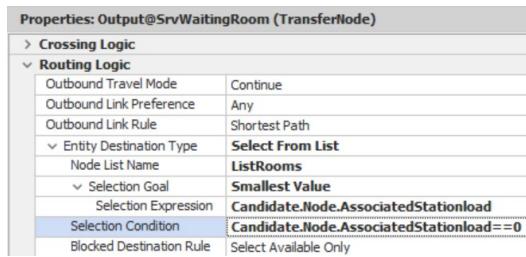
**Table 14.3: Understanding Effect Blocking, Selection Condition, and Input Buffer Size on Routing**

Input Buffer Cap.	Path Travel Capacity	TransferNode’s Blocked Destination Rule	TransferNode’s Selection Condition	Modeling Result
Infinity	1, Infinity	Any Rule	None	All Entities will Wait at the SERVER’s Input Buffer.
Infinity	1, Infinity	Any Rule	Candidate.Node. AssociatedStationload==0	All Entities will wait at the TRANSERNODE (i.e., Waiting Room)
0	1, Infinity	Select Available Only	None	All Entities will wait at the TRANSERNODE
0	Infinity	Prefer Available, Select Any	None	All Entities will wait at the SERVER’s Input Node waiting to go into SERVER.
0	1	Prefer Available, Select Any	None	One additional Entity will wait at the SERVER’s Input Node, while all others will wait at the TRANSERNODE.

**Step 7:** Set all the *Input Buffer* capacities of the rooms back to at least one or “Infinity” to allow one entity to wait in the room. Also, reset **SrvRoom3**’s *Before Processing* and *Processing* add-on process triggers back to **SeizeVet** and nothing, respectively.

**Step 8:** From Table 14.3, relying on blocking the node’s associated station alone may be problematic. Therefore, we will set the **Output@SrvWaitingRoom**’s *Selection Condition* property to be

`Candidate.Node.AssociatedStationLoad == 0`, which says only nodes with no entities routing to them or currently in the server are eligible to be selected.



**Figure 14.9: Forcing Patients to Wait Until a Room is Available**

*Question 5:* For a run of 40 hours, what is the average time in the system and the average wait in the waiting room?

---

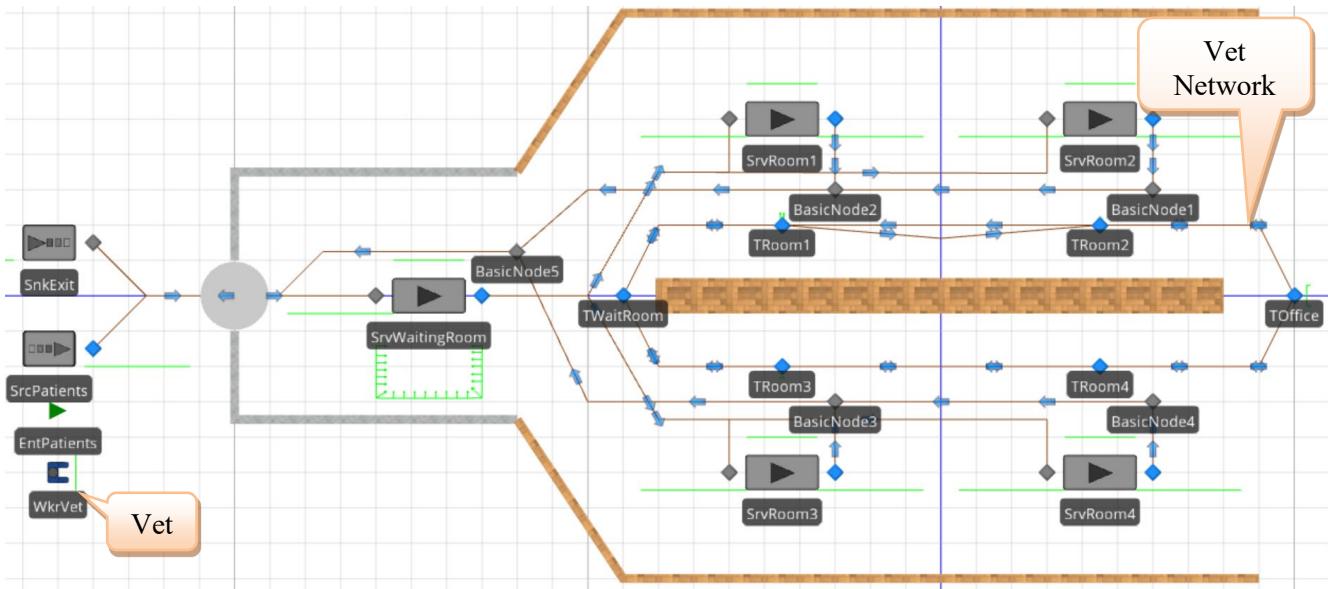
*Question 6:* Does the vet travel to the individual rooms to service the patients?

---

### Part 14.3: Using a WORKER as a Moveable Resource

In this case, even though we modeled the veterinarian as a **WORKER**, it acted like a stationary **RESOURCE** since we did not request that the vet move to the room to service the patient. **WORKERS**, like **ENTITIES** and **VEHICLES**, are dynamic objects that can move throughout a network. Our vet is now being seized and released, constraining the processing of patients rather than assuming four stationary vets in each room. However, the vet is servicing the patients from the **TOffice** node. We do not want to seize the **WkrVet** worker until the vet is in the room.

**Step 9:** Create a separate network for the veterinary staff to move throughout the system. Add five additional **TRANSFERNODES** named **TRoom1**, **TRoom2**, **TRoom3**, **TRoom4**, and **TWaitRoom**. The **TRoom** nodes should be located near the same room as the office in the back, as seen in Figure 14.10. Having a separate network from patients can be advantageous in not causing bottlenecks, etc. However, in the next chapter, we will need them to travel on the same network of paths to allow the **WORKER** to act like a vehicle.



**Figure 14.10: Modified Vet Clinic with Vet Network**

**Step 10:** Connect the nodes using two unidirectional PATHS (i.e., one forward and one backward) since the staff can move in either direction to service the next patient. Use logical distances of 15 meters for each link.<sup>212</sup> The staff will travel on the inner loop.

**Step 11:** Since the WORKER entity acts like a VEHICLE, the vets will utilize the parking queues when they stop at each room and office.<sup>213</sup> By default, a parking queue is automatically placed. However, you cannot orient the queue or place it in a different location. If you wish to improve the animation, then select one of the transfer nodes associated with the Vet network (TRooms or the TOOffice node) and toggle the *Parking Queue* button off. Next, select the ParkingStation.Contents queue from the “Draw Queue” section of the “Attached Animation” ribbon and draw the animated queue very near the node selected choosing the orientation. Repeat the process for the remaining nodes in the Vet network.

**Step 12:** From Figure 14.6 you will notice the *Request Move* property, which can be changed to “ToNode” and then specify the appropriate destination node (e.g., TRoom1 for room one). This will force the WkrVet to move to the TRoom1 TRANSFERNODE before processing. The *Move Priority* property can be used to specify the priority if multiple patients have requested the worker. A tokenized process will be used since the SeizeVet process is the same for all rooms, except the node will change (i.e., TRoom1 to TRoom2).

- Add a new TOKEN named TknVet from the *Definitions* tab that will have a *Node Object Reference* state variable named TStaWhichNode that will allow us to pass in the correct node, as seen in Figure 14.11.

<sup>212</sup> Select all the paths and specify the direction and the logical length at one time or *Object Property Spreadsheet*.

<sup>213</sup> This is for animation purposes since the Vets would disappear when they arrive at their destination and begin service.

Tokens	
TknVet	
Name Object Type	
State Variables	
ReturnValue	Real State Variable
TStaWhichNode	Node Reference State Variable

Figure 14.11: Creating a Specialized Token to Pass in the Move Node

- To create the Tokenized process, select the **SeizeVet** process from the *Processes* tab and specify the **TknVet** as the *Token Class Name* property value, as seen in Figure 14.12. Specify one input argument that allows the user to pass into the process which room needs the service.

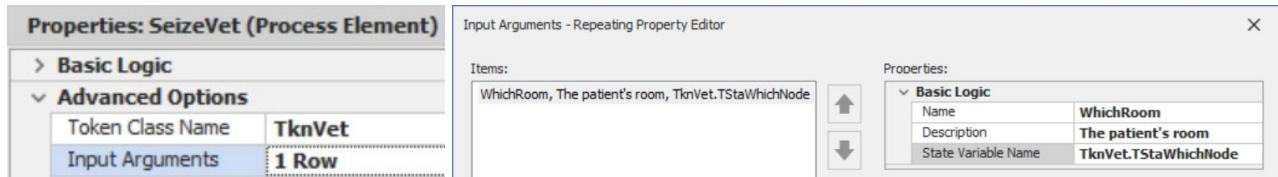


Figure 14.12: Making the SeizeVet Process Tokenized by Passing in one Input Argument

- Next, modify the *Seize* step to specify the *Request Move* to a particular node, and the *Destination Node* will be the **TknVet.TStaWhichNode**. This will force the vet to move to a particular node before processing can begin on the patient, considering the travel time to the room.

Resource Information	
Resource Type	Specific
Resource Name	WkrVet
Selection Goal	PreferredOrder
Request Move	ToNode
Destination Node	TknVet.TStaWhichNode
Move Priority	

Figure 14.13: Modify the Seize to force Vet to Move to the Specified Node

- Finally, on each of the rooms, specify the appropriate Which Room input argument property for the *Before Processing* add-on process trigger, as seen in Figure 14.14 for **SrvRoom1**.

Before Processing		SeizeVet
Input Arguments		
Which Room	TRoom1	
Processing		
After Processing	ReleaseVet	

Figure 14.14: Specify the Correct Node Associated with each Room

**Step 13:** Save and run the model, observing how the veterinarian dynamically moves to each room.

**Question 7:** For 40 hours, what is the average time in the system and the average wait in the waiting room?

**Step 14:** Add a second veterinary staff member by changing the *Initial Number in the System* property under the *Population* property category to two.<sup>214</sup>

**Question 8:** For 40 hours, what is the average time in the system and the average wait in the waiting room?

---

**Question 9:** Is the second veterinary staff member justified, and why?

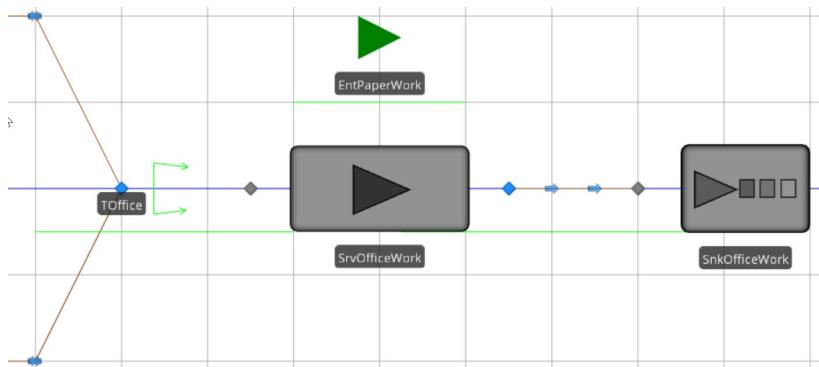
---

#### Part 14.4: Returning to the Office between Patients

One additional modeling concern is that veterinarians need to return to their office after each visit to complete paperwork on the current patient and look over the next patient's chart. All this needs to be done before returning to one of the rooms to service the next patient. So we need to force them to go back to the office and delay for a particular time before returning to service the next patient. Since resources respond to requests to be allocated (i.e., patients trying to seize them), one can model the situation where an entity requests the **WkrVet** at the office so they will return to the office and process that paperwork entity. In the office, the paperwork uniformly takes between four and fourteen minutes.

**Step 1:** Save the current model as a new model and set the number of available veterinaries back to one.

**Step 2:** Insert a **SERVER** named **SrvOffice** near the **TOffice** transfer node and connect it to a **SINK** called **SnkOfficeWork** via a connector as seen in Figure 14.15. Specify the processing time in the **SrvOffice** as Uniform(4,14) minutes.



**Figure 14.15: Modeling the Paperwork**

**Step 3:** Insert a new **MODELENTITY** called **EntPaperWork**. It will be this entity that is processed in the office. Since we want the Vet to do the paperwork before seeing any other patient, let's give the *Initial Priority* of the **EntPaperWork** a value of "2" (recall that the default value is "1"). Then to ensure that this paperwork is given priority, change the *Dynamic Selection Rule* property of the **WkrVet** to "Largest Value First," with the *Value Expression* as *Candidate.Entity.Priority*.<sup>215</sup>

---

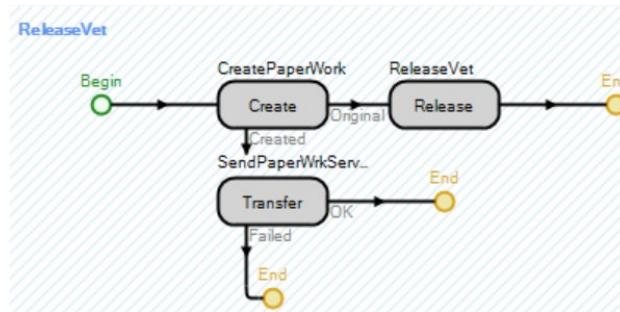
<sup>214</sup> Another option is to insert a new worker named **VetB**. Then a resource list would have to be created to allow entities to seize one from the pool of resources. This modeling approach may be necessary if the two staff members can service different types of patients, different processing times, have a different ordering or work schedule, etc.

<sup>215</sup> The *Dynamic Selection Rule* is used here only to illustrate its use and is strictly not necessary. The only case when the dynamic selection rule is needed is when the Value may be changed while the entity is waiting, such as when a due date is reached.

Ranking Rule	First In First Out
Dynamic Selection Rule	<b>Largest Value First</b>
Value Expression	Candidate.Entity.Priority

**Figure 14.16: Changing the Dynamic Selection Rule of the Worker**

**Step 4:** When the vet finishes at a room, paperwork needs to be created and sent to the office. This paperwork needs to *Seize* the **WkrVet** before another room seizes the vet. Also, as a part of this process, the vet must be released from the room to visit the office. Therefore, we need to modify the **ReleaseVet** process to accomplish all this, as shown in Figure 14.17. Recall that this process is used by all the rooms when patients exit.



**Figure 14.17: Creating and Transferring the Paperwork**

- Use the *Create* step to generate a paperwork entity and a new TOKEN associated with the new entity.

Properties: CreatePaperWork (Create Step Instance)	
Basic Logic	
Create Type	NewObject
Entity Type	EntPaperWork
Number To Create	1

**Figure 14.18: Create Step to Produce an MODELENTITY**

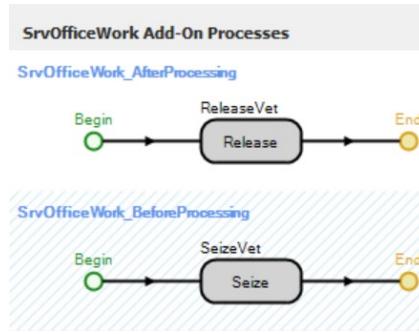
- Use the *Transfer* step to send the paperwork entity to the input buffer station of the office SERVER.<sup>216</sup>

Properties: SendPaperWrkServer (Transfer Step Instance)	
Basic Logic	
From	FreeSpace
To	Station
Station Name	SrvOfficeWork.InputBuffer

**Figure 14.19: Using the Transfer Step to move the New Paper ENTITY.**

**Step 5:** For the **SrvOffice**, create two new add-on process triggers processes for *Before Processing* and *After Processing*.

<sup>216</sup> Note this time we are sending the entity directly to the input buffer station rather than the Input node at the server which will force the paper work to seize the vet without having to be transferred from the input node to the input station.



**Figure 14.20: Requesting and Freeing the Vet from Paper Work at the Office**

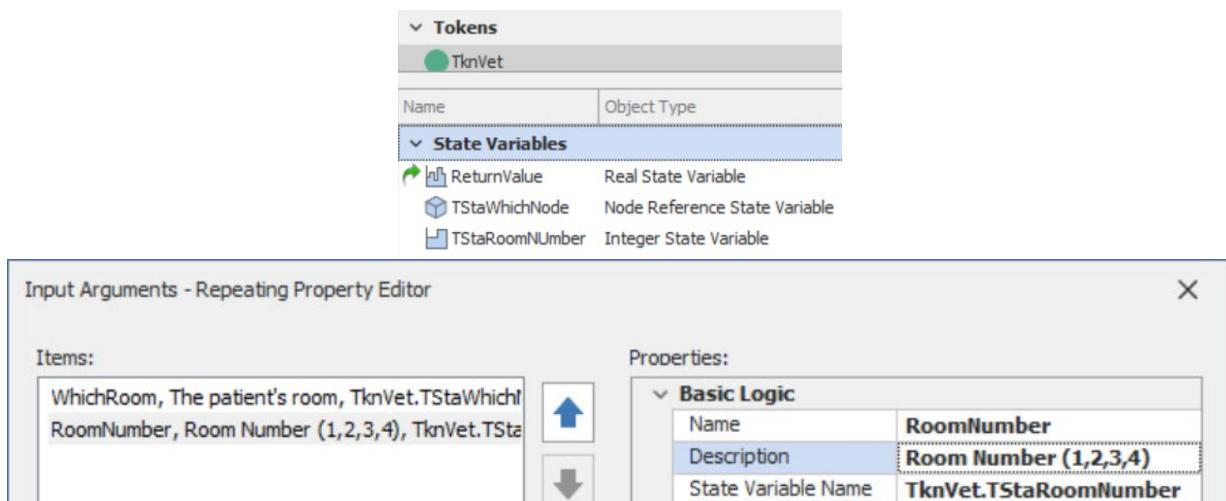
- Copy the *Seize* step from one of the four rooms to request the **WkrVet** for the paperwork. Remember to change the node to visit the **Toffice** node in the *Seize* step.
- Copy the *Release* step from **ReleaseVet** to release the Vet after the paperwork is finished.

**Step 6:** Save and run the model and observe what happens.

*Question 10:* Is the model working as expected, or do you observe any issues?

**Step 7:** The issue seems that the vet will process a second patient before heading back to the office. Essentially, the vet processes the previous patient's paperwork, but it is difficult to determine which paperwork went with which patient since all the patients and paperwork entities are green. To alleviate the situation, add three additional symbols for both the **EntPatients** and **EntPaperWork**, giving each one of the additional symbols a different color and making sure the symbols for each entity type are the same color (i.e., 0<sup>th</sup> symbol is green for both patient and paperwork, 1<sup>st</sup> symbol is red for both, 2<sup>nd</sup> symbol is yellow, and the 3<sup>rd</sup> symbol is blue).

**Step 8:** Patients will be assigned a color based on the room they are serviced in (i.e., patients will be green in Room 1, red in Room 2, yellow in Room 3, and finally blue in Room 4). Since this will be done when the patient goes into processing, we need to add an additional input argument to the **SeizeVet** process to specify which picture to use. First, add an **INTEGER STATE** variable named **TstaRoomNumber** to our **TknVet** and then specify the second input argument of the **SeizeVet** process to be the room number as seen in Figure 14.21.



**Figure 14.21: Adding a new Input Argument to the Tokenized SeizeVet Process**

**Step 9:** Next, add an *Assign* step to **SrvRoom2\_BeforeProcessing** before the *Seize* step that assigns a new value of “1” to the `ModelEntity.Picture` state variable.

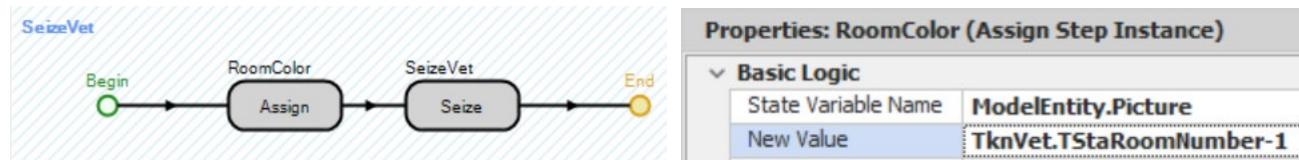


Figure 14.22: Changing the Patient Color based on the Room

**Step 10:** Modify the *Before\_Processing* for each of the rooms and specify “1”, “2”, “3”, and “4” for the Room Number input argument for **SrvRoom1**, **SrvRoom2**, **SrvRoom3**, and **SrvRoom4** respectively.

▼ Before Processing	SeizeVet
▼ Input Arguments	
Which Room	TRoom1
Room Number	1
Processing	
After Processing	
ReleaseVet	

Figure 14.23: Passing the Room Number for SrvRoom1

*Question 11:* Why are we subtracting one from the room number in the *Assign* step?

---

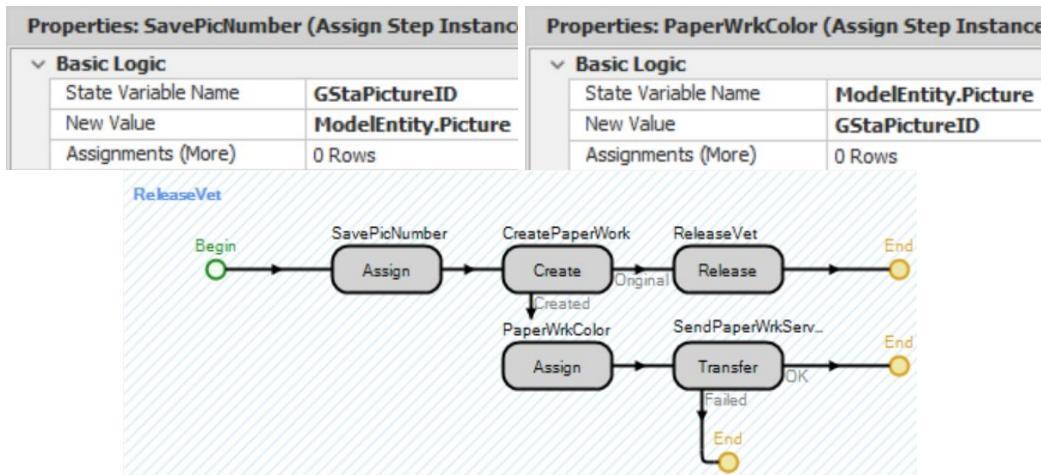
**Step 11:** Save and run the model, observing the entities as they enter the various rooms.

**Step 12:** The patient entities are changing color based on the room they enter. The paperwork entity now needs to reflect the color of the patient that created it. Process steps like *Create* and *Search* cause new tokens to be created that are now associated with either the new entity created in the *Create* step or the objects found in the *Search* step. Therefore, state variables associated with the original token’s object (i.e., the `ModelEntity.Picture`) are not accessible to the new objects. One common approach to copy states from one entity to another is to create a global (i.e., `MODEL`) state variable that is accessible by all objects of the model.<sup>217</sup> Add a new `INTEGER DISCRETE STATE` variable named **GStaPictureID** from the *Definitions* tab.

**Step 13:** Next, add an *Assign* step before the *Create* step in the **ReleaseVet** process that assigns the **GStaPictureID** to the `ModelEntity.Picture` of the patient entity that is creating the paper work. Then insert another *Assign* step before the *Transfer* which is associated with the newly created paper work entity that assigns the model state variable **GStaPictureID** to the `ModelEntity.Picture` state variable as seen in Figure 14.24.

---

<sup>217</sup> Instead of creating a new Object in the *Create* step, a copy of the associated object could be created which would inherit all the properties and states of the creating object. This approach could have been used in this case but the statistics associated with **EntPatients** could not be used since this would include the paper work entities as well since they would be the same entity type. Also we could have created a tokenized **ReleaseVet** process that would take in the room number and therefore would know the color that needed to be set.



**Figure 14.24: Copying the Value from One ModelEntity to another ModelEntity**

**Step 14:** Save and run the model, observing the entities as they enter the various rooms and the paperwork.

**Question 12:** Can you see the different paperwork entities being created and how the vet always lags behind by one?

---

## Part 14.5: Zero-Time Events

The issue seems to be that the vet essentially processes the previous patient's seizure even though the paperwork has a higher priority. A discrete-event simulation executes the simulation through an event calendar that is ordered by the time of the event. Zero-time events are events that occur at the exact same time.<sup>218</sup> The *Seize* step for the paperwork entity happens simultaneously with the *Release* step for the room. If the release step occurs before the *Seize* step occurs, the vet may serve a patient in a different room. Ordering of zero-time events can become critical in the execution of the model, and the modeler won't know which is done first without some detective work.

A great tool for understanding what is happening during your simulation is through the use of the “*Model Trace*.” The “*Model Trace*,” initiated from the “*Run*” tab, is the most complete description of what is happening. However, in this case, you want to trace the behavior of a process. You do this by selecting the step in the *Processes* tab and clicking the breakpoint button in the *Process Tables* ribbon, which will place the breakpoint symbol adjacent to the step<sup>219</sup>. Now, in the “*Facility*” tab, select the *Model Trace* button, which displays the trace in its own window.

**Step 1:** Click on the *Create* step in the **ReleaseVet** process and select the breakpoint button. The *Create* step should now have the breakpoint symbol beside it.

**Step 2:** Click the *Model Trace* button on the *Run* tab and then run the model via the *Fast-Forward* button.

**Step 3:** After the simulation “breaks”, click the *Step* button a couple of times. (You can recognize the break because the line colored red describes the breakpoint.) When you click the *Step* button, a series of actions

---

<sup>218</sup> There are dozens of events that can occur at the same time and can become an issue especially when dealing with allocations of resources, workers, and vehicles.

<sup>219</sup> You can also right-click the step and select the “breakpoint”.

are shown, colored in yellow, representing that simulation step. You may need to scroll back up to see the breakpoint statement.

**Step 4:** Figure 14.25<sup>220</sup> is a printout of all the events that have occurred around the breakpoint.

Trace - Tracing to Q:\Shared drives\SIMIO Book\New Book 2023\Chapter14\Chap 14-4_Model_trace.csv							
D...	Time (hours)	Entity	Object	Process	Token	Step	Action
5...	0.292329399731871	EntPatients.67	Model	ReleaseVet	3	[Create] CreatePaperWork	Stopped at breakpoint.
							Creating '1' new object(s) of entity type 'EntPaperwork'.
							Created object EntPaperwork.70.
		EntPaperwork.70	Model	ReleaseVet	5	[Assign] PaperWkColor [Transfer] SendPaperWkServer	Assigning state variable 'EntPaperwork.70.Picture' the value '0'. Previous value was '0'.
							Entity 'EntPaperwork.70' transferring from '[FreeSpace] Model' into station 'SrvOfficeWork.InputBuffer'.
							Object 'EntPatients.67 attempting to release '1' resource(s) of resource type '[Specific] WkrVet'.
		EntPatients.67	Model	ReleaseVet	3	[Release] ReleaseVet	Object 'EntPatients.67 released '1' capacity unit(s) of resource 'WkrVet[1]'.
							Changing from AutoState 'Busy' to 'Starved' on resource 'WkrVet[1]'.
							Assigning state variable 'WkrVet[1].ResourceState' the value '0'. Previous value was '1'.
5...	0.292329399731871	EntPatients.67	WkrVet[1]	OnCapacityRel...	10	[Begin] [Assign] ResourceState [Assign] OnCapacityReleasedAss... [Assign] OnCapacityReleasedAss... [SetNode] ToClearDestination [Fire] RemainInPlaceEndedEvent	Process 'WkrVet[1].OnCapacityReleased' execution started.
							Assigning state variable 'WkrVet[1].ResourceState' the value '0'. Previous value was '0'.
							Assigning state variable 'WkrVet[1].OnEvaluatingRiderReservation.Enabled' the value '1'. Previous value was '0'.
							Assigning state variable 'WkrVet[1].OnEvaluatingRiderReservation.ReturnValue' the value '1'. Previous value was '0'.
							Assigning state variable 'WkrVet[1].OnEvaluatingSeizeRequest.Enabled' the value '1'. Previous value was '0'.
							Assigning state variable 'WkrVet[1].OnEvaluatingSeizeRequest.ReturnValue' the value '1'. Previous value was '0'.
							Assigning state variable 'WkrVet[1].OnEvaluatingVzveRequest.Enabled' the value '1'. Previous value was '0'.
							Assigning state variable 'WkrVet[1].OnEvaluatingVzveRequest.ReturnValue' the value '1'. Previous value was '0'.
							Assigning state variable 'WkrVet[1].OnEvaluatingRiderAtpDup.Enabled' the value '1'. Previous value was '0'.
							Assigning state variable 'WkrVet[1].OnEvaluatingRiderAtpDup.ReturnValue' the value '1'. Previous value was '0'.
5...	0.292329399731871	WkrVet[1]	WkrVet[1]	OnListingNode	0	[Wait] UntilRemainInPlaceEnded [Fire] ReleasedEvent [End]	Entity 'WkrVet[1]' assigned destination node 'No Destination'.
							Wait for event(s) ended. Releasing token from Wait step.
							Firing event 'WkrVet[1].Released'.
5...	0.292329399731871	EntPatients.67	WkrVet[1]	OnCapacityRel...	10	[End]	Process 'WkrVet[1].OnCapacityReleased' execution ended.
							Resource 'WkrVet[1]' checking allocation queue which contains '2' item(s).
							Object 'EntPatients.67' checking availability to seize '1' resource(s) of resource type '[Specific] WkrVet'.
5...	0.292329399731871	EntPatients.68	Model	SeizeVet	4	[Seize] SeizeVet	Object 'EntPatients.68' checking availability to seize '1' capacity unit(s) of resource 'WkrVet[1]'.
							Object 'EntPatients.68' found '1' currently available capacity unit(s) of resource 'WkrVet[1]'.
							Object 'EntPatients.68' seized '1' capacity unit(s) of resource 'WkrVet[1]'.

**Figure 14.25:** Sample Trace showing the Ordering of the Zero-Time Events

**Step 5:** Now, you should read down the trace, noting the entity, the object, the step, and the action. Here, we see that the paperwork entity is created (**EntPaperwork.70**), the picture variable is assigned the value of 0, and that entity is transferred to the “*SrvOffice.InputBuffer*.” Next, the patient entity (**Entpatients.67**) releases one unit of capacity of the vet. Scrolling down through the other zero-time events, you will see that the vet will then respond to a request from another room first because the paperwork entity does not request one unit of capacity from the vet until near the very end of the current time.<sup>221</sup> All this is done at the same simulation time, but the ordering is important.

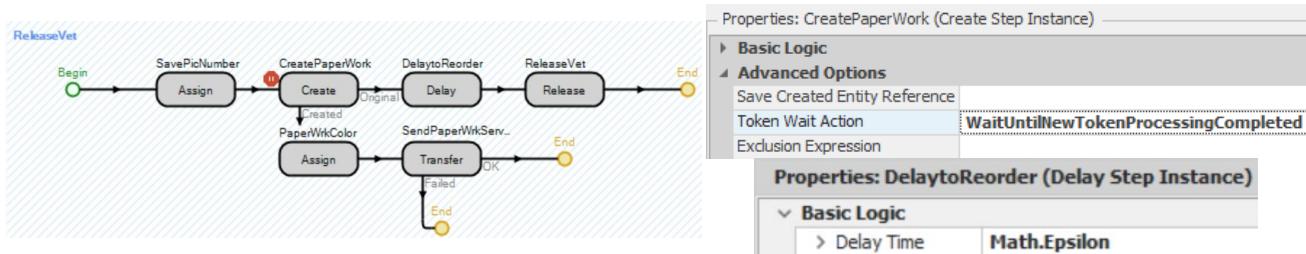
**Question 13:** Can you now see that the *Release* step is done before the office *Seize* step?

**Step 6:** We need the release to happen after the *Seize* step, which implies that the concurrent time events need to be reordered. To accomplish a reordering of events, you can insert a *Delay* step before the release of the vet (see Figure 14.26) that will delay the current token (representing the patient) a very small amount of time by specifying the delay time to be `Math.Epsilon`.<sup>222</sup> Also, change the *Token Wait Action* of the *Create* step to **WaitUntilNewTokenProcessingCompleted**.

<sup>220</sup> The trace is automatically saved to a CSV file as the model is running and can be imported into Excel for further analysis and manipulation if necessary. You can also filter each column by selecting a row and the right clicking on the column heading.

<sup>221</sup> It may be necessary to click the *Step* button again to get the next set of zero time events owing to the number of events that occur.

<sup>222</sup> Math.Epsilon represents the smallest real value number greater than zero. *Delay* steps will actually interpret the number as zero, but it forces the event to be put at the end of the current event calendar. If the delay time is specified as zero, SIMIO will not place the event on the calendar thus not changing the order at all. In the commentary, another solution to the ordering is presented.



**Figure 14.26: Reordering Zero Time Events by Delaying an Epsilon Amount of Time**

**Step 7:** Save and run the model again performing a few “Steps” after the breakpoint which yields the trace in Figure 14.27. As you can see, the delay causes all the steps following it to become late priority events, meaning they will be placed at the end of the current time. This allows the paperwork to seize the vet first, which then allows the vet to select it based on its priority.

Time (Hours)	Entity	Object	Process	Token	Step	Action
					[Begin]	Process 'ReleaseVet' execution started.
					[Assign] SavePicNumber	Assigning state variable 'Model.GStaPictureID' the value '0'. Previous value was '0'.
					[Create] CreatePaperWork	Stopped at breakpoint. Creating '1' new object(s) of entity type 'EntPaperwork'. Created object 'EntPaperwork_70'.
	EntPatients.67	Model	ReleaseVet	3	[Assign] PaperWkColor	Assigning state variable 'EntPaperwork_70.Picture' the value '0'. Previous value was '0'.
					[Transfer] SendPaperWkServer	Entity 'EntPaperwork_70' transferring from '[FreeSpace] Model' into station 'SrvOfficeWork.InputBuffer'.
	EntPatients.67	Model	ReleaseVet	3	[Create] CreatePaperWork	Token waiting at Create step for '1' other processes to complete.
	SrvRoom1	OnEnteredProcessing		2	[Execute] AfterProcessingAddOnProcess	Token waiting at Execute step for '1' other processes to complete.
					[Begin]	Process SrvOfficeWork.OnEnteredInputBuffer execution started.
	EntPaperwork_70	SrvOfficeWork	OnEnteredInputBuffer	6	[EndTransfer] IntoInputBuffer	Entity 'EntPaperwork_70' ending transfer into station 'SrvOfficeWork.InputBuffer'.
					[Seize] Server	Object 'EntPaperwork_70' waiting to seize '1' resource(s) of resource type '[ParentObject] SrvOfficeWork'.
					[End]	Process ReleaseVet execution ended.
	EntPatients.67	Model	ReleaseVet	3	[Delay] DelayToReorder	Delaying token for 'Epsilon' Hours until time '0.292329399731871' Hours. Scheduling end of delay as a late priority event on current events calendar.
	-	-	-	0	-	Resource 'SrvOfficeWork' checking allocation queue which contains '1' item(s).
0.292329399731871...						Object 'EntPaperwork_70' checking availability to seize '1' resource(s) of resource type '[ParentObject] SrvOfficeWork'.
						Object 'EntPaperwork_70' checking availability to seize '1' capacity unit(s) of resource 'SrvOfficeWork'.
	EntPaperwork_70	SrvOfficeWork	OnEnteredInputBuffer	6	[Seize] Server	Object 'EntPaperwork_70' found '1' currently available capacity unit(s) of resource 'SrvOfficeWork'.
	-	-	-	0	-	Object 'EntPaperwork_70' found '1' currently available resource(s) of resource type '[ParentObject] SrvOfficeWork'.
	EntPaperwork_70	SrvOfficeWork	OnEnteredInputBuffer	6	[Seize] Server	Resource 'SrvOfficeWork' selecting object 'EntPaperwork_70' waiting at rank position '1' in allocation queue which contains '1' item(s).
	-	-	-	0	-	Object 'EntPaperwork_70' seized '1' capacity unit(s) of resource 'SrvOfficeWork'.
						Changing from AutoState 'Starved' to 'Busy' on resource 'SrvOfficeWork'.
						Assigning state variable 'SrvOfficeWork.ResourceState' the value '1'. Previous value was '0'.
	EntPaperwork_70	SrvOfficeWork	OnEnteredInputBuffer	6	[Seize] Server	Object 'EntPaperwork_70' completed seize of '1' resource(s) of resource type '[ParentObject] SrvOfficeWork'.
						Executing process SrvOfficeWork.BeforeProcessing.
						Process SrvOfficeWork.BeforeProcessing execution started.
	-	-	-	0	-	Ignoring attempt to execute disable process 'WkrVet[1].OnNewSeizeRequest'.
						Object 'EntPaperwork_70' checking availability to seize '1' resource(s) of resource type '[Specific] WkrVet'.
						Object 'EntPaperwork_70' checking availability to seize '1' capacity unit(s) of resource 'WkrVet[1]'.
						Object 'EntPaperwork_70' unable to seize '1' capacity unit(s) of resource 'WkrVet[1]'. Insufficient capacity available.
						Object 'EntPaperwork_70' found '1' currently available resource(s) of resource type '[Specific] WkrVet'.

**Figure 14.27: Reordered zero-time events**

**Step 8:** Remove the break point by selecting the *Create* step and toggling the Breakpoint button. Now run the model and observe what happens.

**Question 14:** Is the model working as expected?

## Part 14.6: Handling Multiple Vets

In the previous section, there were multiple vets that could process patients. Like other dynamic objects MODELENTITIES and VEHICLES, multiple WORKERS can be created at the start of the simulation or during the simulation run. Let’s add two more vets to help in the clinic.

**Step 1:** Select the **WkrVet** and change the *Initial Number in System* property to “3” under the *Population* category.

**Step 2:** Change the **SrvOffice** capacity to “Infinity” so each of the vets can service their own paperwork.

**Step 3:** Save and run the model observing the vets and paperwork.

**Question 15:** Did the vet that processed the patient always process the correct paper work?

**Step 4:** The difficulty is that the paperwork entity just requested a vet, and the closest one who was not busy went to the office to perform paperwork (or was already sitting at the office), which was not what was intended. We wanted the vet who performed the service to also complete the paperwork. Therefore, the paperwork entity needs to request the same vet as the patient.<sup>223</sup> First, select the MODELENTITY in the [Navigation] panel and insert a new MODELENTITY DISCRETE INTEGER STATE variable named **EStaResourceID**, which will be used to store the ID of the resource to request for every paperwork entity.

**Step 5:** Transition back to the **Model** and insert a new model DISCRETE INTEGER STATE variable named **GStaModelResID**. This variable will be used to pass the resource ID that was seized by the patient to the paperwork entity in a similar fashion as the picture number that was passed.

**Step 6:** Next, we need to set the global (i.e., model) variable (**GStaModelResID**) with the vet the current patient has seized before we create the paperwork entity and then set the paperwork entity's **EStaResourceID** variable the value of the model variable. Inside the **ReleaseVet**, modify the first *Assign* step (i.e. **GStaPictureID**) to include another assignment which sets **GStaModelResID** a value of **ModelEntity.SeizedResources.LastItem.ID**.<sup>224</sup> Then, modify the second *Assign* step on the "Created" path (i.e., **PaperWorkColor**) to set the **ModelEntity.EStaResourceID** variable a value of **GStaModelResID** as seen in Figure 14.28.

Basic Logic	
State Variable Name	New Value
GStaModelResID	ModelEntity.SeizedResources.LastItem.ID

Basic Logic	
State Variable Name	New Value
ModelEntity.EStaResourceID	GStaModelResID

Figure 14.28: Passing Seized Resource ID to the Newly Created PaperWork Entity

**Step 7:** Modify the *Seize* step in the **SrvOffice\_BeforeProcessing** process to request a particular Vet that needs to perform the paperwork based on the state variable **ModelEntity.EStaResourceID**. Under the *Advanced Options*, specify the *Selection Condition* property to be **Candidate.Worker.ID == ModelEntity.EStaResourceID** as seen in Figure 14.29. Recall the **CANDIDATE** is a wild card designator and will match all the **WkrVets** that are currently in the system. The *Selection Condition* property has to evaluate to "True" for the **WkrVet** to be in the pool of potential ones to be seized and in this example all **WkrVets** will be eliminated for consideration except the one whose ID matches the **EStaResourceID** value.

---

<sup>223</sup> All objects created in the simulation receive a unique identifier (ID) which can be used. For example **EntPatients.61** is a patient whose ID is 61.

<sup>224</sup> The vet will be the last resource seized since the patient had to first seize the server before it was allowed to go into processing and then request the vet. See Table 13.2 for more information on the **SeizedResources** property of the entity.

Resource Information	
Resource Type	Specific
Resource Name	<b>WkrVet</b>
Selection Goal	PreferredOrder
Request Move	<b>ToNode</b>
Destination Node	<b>TOffice</b>
Move Priority	
Advanced Options	
Quantity Type	Specific
Number Of Resources	1
Units Per Resource	1
Selection Condition	<b>Candidate.Worker.ID == ModelEntity.EStaResourceID</b>

**Figure 14.29: Selecting Only the Correct Vet for the Paper Work**

**Step 8:** Save and run the model observing the vets and paperwork.

*Question 16:* Did the vet that processed the patient always process the correct paper work?

---

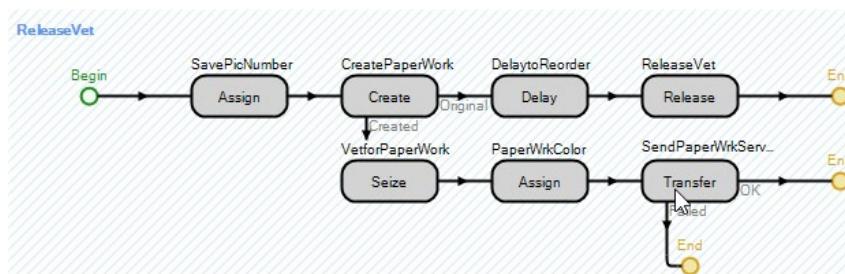
### Part 14.7: Commentary

- Dynamic resources offer considerable modeling capability. In this chapter, we utilized processes to seize and release the work stating it offers the most control over the process. However, in this case we could have utilized secondary resources to accomplish the same task of the two processes used to seize and release the vet. Figure 14.30 displays *For Processing* property *Resources for Processing* repeat group values under the *Secondary Resources* section for the **SrvRoom1** that would seize the vet and wait for it to arrive before processing.

Resource Information	
Resource Type	Specific
Resource Name	<b>WkrVet</b>
Selection Goal	Preferred Order
Request Move	<b>To Node</b>
Destination Node	<b>TRoom1</b>
Required Quantity & Constraints	
Advanced Options	

**Figure 14.30: Specifying the Seizing and Release of the Vet Using Secondary Resources**

- We fixed the zero-time event by reordering the concurrent events using a delay step. Another solution would have been to seize the vet before transferring, as seen in Figure 14.31. The “Original” token waits until the “Created” token reaches the end of the current process. Therefore, the seize will happen before the release in the zero-time events.



**Figure 14.31: Seizing the Vet before Transferring the Paperwork**

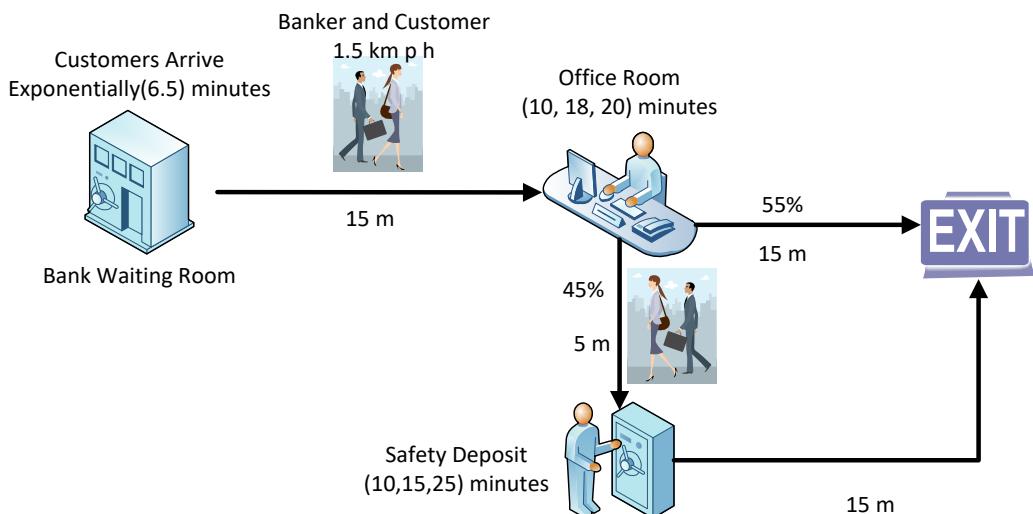
- Decision-making objects offer a number of modeling options especially as they relate to resources, transporters, and workers. Because there is so much flexibility, it can confound the choices. However, by working with these objects, you begin to develop some modeling patterns that seem more appropriate than others.
- While it is our intent to use as many modeling approaches as we can, it will still be necessary for you to supplement our efforts. Features such as the breakpoints, trace, and watch can be extremely helpful in learning how SIMIO is behaving. It's hard to modify a behavior unless you understand the existing behavior.
- Unfortunately, the behavior of workers and vehicles is quite complex. Because of the complex behavior, these objects could be called “heavyweight.” In some object-oriented designs, heavyweight objects are composed of “lightweight” objects which are less complex. However, that is not the case with SIMIO, so we will continue to investigate and employ the complexity of these objects.

# Chapter 15

## Adding Detail to Service: A Bank Example

We have previously modeled a moveable resource as a worker. Entities would request the resource to visit a specific node in the `Seize` step, forcing the worker to move to the node before allowing the entity to enter service. `VEHICLES` can also be seized and used as resources in the exact same way. The `WORKER` object is a specialized `VEHICLE` that can transport entities as well as be seized and released as a resource.<sup>225</sup> The modeling concepts relative to moveable resources used in this chapter will apply to vehicles or workers.

A small bank has several loan officers who assist customers with various loans and other account information, as seen in Figure 15.1. Customers arrive and wait for an available loan officer. The loan officer goes to the waiting room and escorts (i.e., picks up) the customer back to the office to help the customer. After this process is complete, some customers require a visit to their safety deposit boxes, which requires the loan officer to escort them to the vault area and assist the customer again. The loan officer can be modeled using a worker or a vehicle.



**Figure 15.1: Bank Example where Worker Picks up Customers as well as Services Them**

### Part 15.1: Using a Worker as a Resource and a Vehicle

One advantage of using a worker as a moveable resource is the worker can also be used as a vehicle. For example, the Vet or a technician may go to the waiting room and escort (i.e., pick up) the patient to the room where the Vet sees the patient (i.e., the Vet is seized). That kind of transportation is being done in the bank.

**Step 1:** Create a new model.

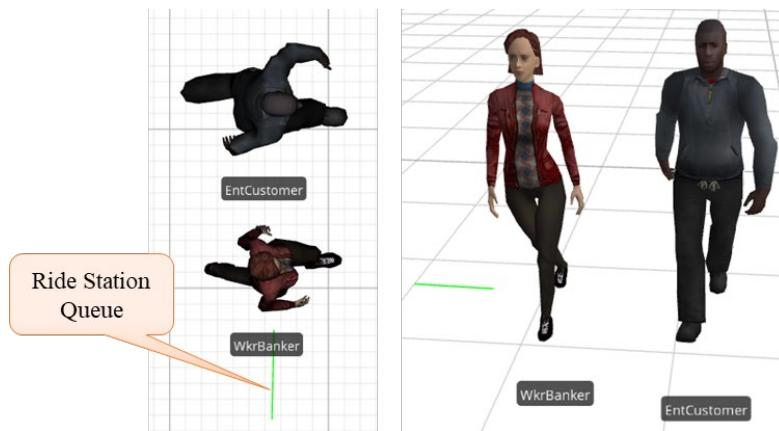
**Step 2:** Insert a `SOURCE` named `SrcWaitingRoom`, a `SERVER` named `SrvRoom`, and a `SINK` named `SnkExit`.

<sup>225</sup> A `WORKER` object must work “on demand” and has no routing type, meaning they don’t follow a specific sequence.

- Connect the output of the **SrcWaitingRoom** to the input of the **SrvRoom** via a 15-meter path and then the output of the room to the **SINK (SnkExit)** by another 15-meter path.<sup>226</sup>
- Customers arrive exponentially with an interarrival time of 6.5 minutes.
- The capacity of the **SERVER** should be one.
- Each customer takes between 10 and 20 minutes to be seen, with most of them taking 18 minutes.

**Step 3:** Insert a new MODELENTITY named **EntCustomer** and a WORKER<sup>227</sup> named **WkrBanker** into the model. The customers and bankers travel at a constant rate of 1.5 km per hour or 1500 meters per hour.

- For the **EntCustomer** entity, add four additional symbols and change each of the five symbols to be represented by one of the *People\Animated* symbols in the *Library*<sup>228</sup>. If you utilize just the “static” “people” symbols, the people will “skate” across the paths. The animated symbols will allow the people to physically walk, run, and/or skip. For the **EntCustomer**, set the *Animation→Random Symbol* to “True” so it will randomly select one of the five people.
- Change the symbol of the **WkrBanker** to one that is more representative using one of the animated people as well. To simplify the presentation, we delete all ten of the alternative symbol states.
- For the **WORKER**, move the animated ride queue (Ride Station Queue)<sup>229</sup> so it is beside the **WkrBanker** instead of on top for a normal **VEHICLE**, as seen in Figure 15.2. Notice that the ride station queue is automatically given the option of “Match Attached Animation Speed,” meaning the “rider” will walk at the same speed as the worker.
- Also, for the worker, set the *Initial Node (Home)* to **Output@SrcWaitingRoom** and set the *IdleAction* to **Park at Home**.
- Now, for the **WkrBanker**, set the *Park While Busy* property under the *Resource Logic* section to “True” to force the banker to park when it is utilized as a resource.
- Make sure the **SOURCE** creates the **EntCustomer** MODELENTITIES.



**Figure 15.2: Two and Three-Dimensional Versions of a Banker and Customer**

**Step 4:** Save and run the model, making sure the entities are flowing correctly.

---

<sup>226</sup> Select both paths and change the *Drawn To Scale* property to **False** and the *Logical Length* to be 15 meters simultaneously.

<sup>227</sup> One can also use a **Vehicle** as a transporter and resource.

<sup>228</sup> Symbols in the Library can be selected based on “Domain”, “Type”, and “Action”.

<sup>229</sup> By holding down the *Shift* key while in 3-D mode will allow you to move the queue up and down. Once you have it on the floor, switch back to the 2-D view to position it beside the banker.

*Question 1:* Does it generate random people symbols arriving at the bank, which are physically walking, and what is the banker currently doing?

---

**Step 5:** In the next step, the **WkrBanker** should pick up the customers from the waiting room and deliver them to the room. Change the outgoing transfer node (**Output@SrcWaitingRoom**) at the **SrcWaitingRoom** to require a transporter, as seen in Figure 15.3. Change the *Ride On Transporter* property to “True,” choose a specific *Transporter Type*, and specify the **WkrBanker** as the vehicle requested.

Transport Logic	
Ride On Transporter	Always
Transporter Selection	
Transporter Type	Specific
Transporter Name	<b>WkrBanker</b>
Reservation Method	Reserve Closest
Selection Goal	Preferred Order
Selection Condition	

**Figure 15.3: Changing the Node Logic to Request a Transporter**

**Step 6:** Make the path between the **SrcWaitingRoom** and the **SrvRoom** bidirectional so the banker can move between the two locations. Generally, two unidirectional paths are better to prevent blockage.

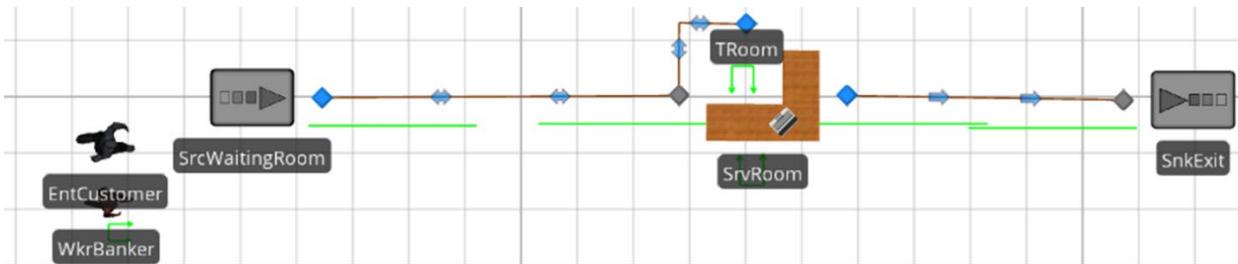
**Step 7:** Save and run the model, making sure the WkrBanker moves the entities to the room.

*Question 2:* What did you observe (i.e., does the **WkrBanker** stay with the **EntCustomer** while they are being processed)?

---

The **WkrBanker** is currently operating as a **VEHICLE** only that transports the customers from the waiting room to the office. It does not necessarily stay with the customer while it is being processed but continues back to pick up the next customer and move them to the room.

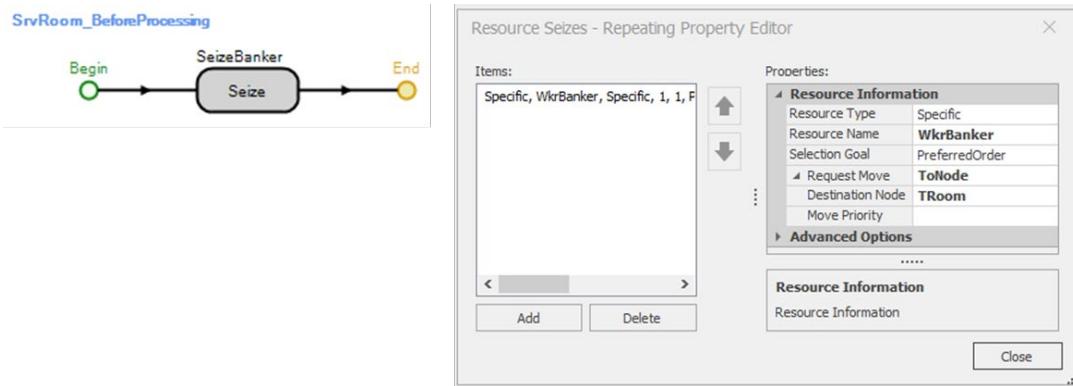
**Step 8:** Insert a new **TRANSFERNODE** named **TRoom**<sup>230</sup> above the **SrvRoom** and connect it via a one-meter bidirectional path to the input of the **SrvRoom**, as seen in Figure 15.4. The **TRoom** node will serve as the location where the worker will serve the customer. This node could also serve as the home node for the worker if the worker is idle. So, for **WkrBanker**, modify the *Initial Node (Home)* to **TRoom**.



**Figure 15.4: Model that Uses Vehicles as both a Resource and a Transporter**

<sup>230</sup> This node will be used as the node to visit in the *Seize* block. Note, the node that a worker or vehicle is dropping off entities cannot be the same node they need to visit. If the same node is specified, the model will result in a deadlock situation with the Worker/Vehicle stuck at the node and the Entity frozen in the **InputBuffer** and not able to proceed into processing.

**Step 9:** In the *Before Processing* add-on process trigger of the **SrvRoom SERVER**, use a *Seize* step that will seize the **WkrBanker** and request a visit to the **TRoom** node (see Figure 15.5 for more information).<sup>231</sup>



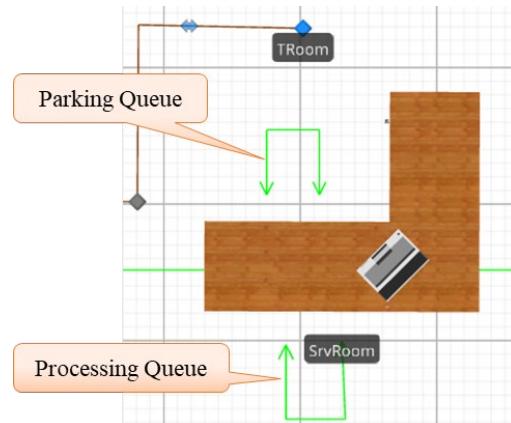
**Figure 15.5: Requesting a Banker for a Customer to be seen at the TRoom Node.**

**Step 10:** In the “*After Processing*” add-on process trigger of the **SrvRoom SERVER**, use a *Release* step that releases the **WkrBanker** once it has seen the customer.<sup>232</sup>

**Step 11:** Save and run the model, observing what happens.

**Question 3:** Did the **WkrBanker** transport the customer to the room and then move into place to service the entity before heading back to get the next customer?

**Step 12:** When the customer seizes the banker at the office, it is automatically shown in the parking queue of the **TRoom TRANSERNODE**, and the customer is shown automatically in the processing station contents queue above the server **SrvRoom**. We will improve the animation to have the customer and banker face each other across a desk, as seen in Figure 15.6.



**Figure 15.6: Modifying the Animation of the Banker’s Office**

<sup>231</sup> Allocation as a Resource (i.e., responding to seizures) takes priority over handling ride requests from entities. If there is an entity that has requested a ride and one that has requested its service as resource, the vehicle will handle the resource request first.

<sup>232</sup> We could have used the Secondary Resources section *Resource for Processing* that will seize and release the worker. However, in the next sections we will make modifications that cannot be done with Secondary Resources.

- Select the **SrvRoom** server and change the symbol to a desk by selecting the “*DeskI*” symbol in the *Library/Furniture* section.
- Move the processing contents queue below the desk and change its alignment to *Oriented Point*. Make sure the oriented points are facing the desk, as seen in the figure.
- Next, select the **TRoom** transfer node and turn off the automatic parking queue by toggling the *Parking Queue* button in the *Appearance→Attached Animation* section. From the *Draw Queue* dropdown, insert a *Parking Station*.*Contents* with *Oriented Point* alignment.<sup>233</sup>

**Step 13:** Save and run the model, observing what happens when they arrive at the office.

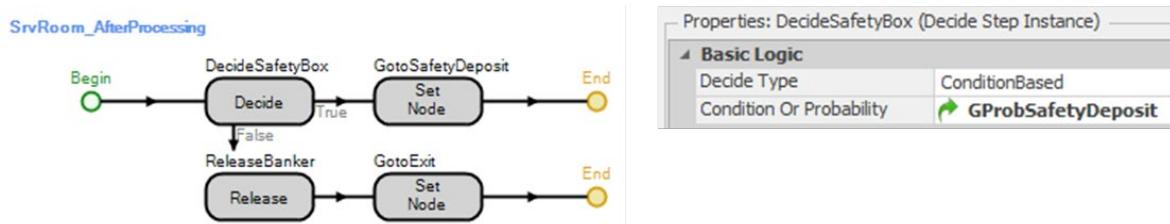
## Part 15.2: Having the Banker Escort the Customer to the Safety Deposit Box Room

In this bank, 45 percent of customers need to visit their safety deposit box after seeing the banker.<sup>234</sup> A similar modeling method as used previously to route patients to an X-ray machine with the assistance of the Vet will be used. The **WkrBanker** resource will not be released if the customer needs to visit the safety box.

**Step 1:** Insert an additional SERVER named **SrvSafetyDeposit**.

- Set the capacity of the SERVER to one since there is only one vault area.
- The processing time for a customer to retrieve their safety deposit box and perform any actions follows a Pert distribution with a min, most likely, and maximum values of 10, 15, and 25 minutes, respectively.
- Connect the output of the **SrvRoom** to the input of the **SrvSafetyDeposit** via a five-meter PATH.
- Connect the output of the **SrvSafetyDeposit** to the **SnkExit** via a 15-meter logical PATH.

**Step 2:** Modify the **SrvRoom\_AfterProcessing** add-on process trigger to model the banker behavior when the customer needs to be escorted to the safety deposit box 45% of the time. From the *Definitions* tab, add a new REAL PROPERTY named **GProbSafetyDeposit** and set the default to 0.45. Insert a *Decide* step (as seen in Figure 15.7) that is probabilistic based with **GProbSafetyDeposit** as the expression.



**Figure 15.7: Using Set Node to Send Customers to the Exit or Vault Machine**

**Step 3:** If a visit to the safety deposit box is needed (i.e., the true branch), insert a *Set Node* step that will set the destination node of the entity to the input at **SrvSafetyDeposit**.<sup>235</sup> If not visiting the safety deposit box (i.e., the false branch), release the **WkrBanker** and then set the destination node to the input at the

<sup>233</sup> You will need to orient the desk by selecting the desk. While holding down the *Ctrl* key, grab one of the corners and rotate the desk around to the correct orientation.

<sup>234</sup> The percentage is deliberately high to determine if the model is behaving correctly.

<sup>235</sup> The *Set Node* step is identical to setting the *Entity Destination* property of a *TransferNode* (i.e., “Continue”, “Specific”, “By Sequence”, or “Select from List”). The *Destination Type* can be a “Specific Node”, choose the node in the next sequence (i.e., “By Sequence”) or a complicated expression.

**SnkExit.**<sup>236</sup> See Figure 15.8 for the property details of the two Set Node process steps. When the entity enters the TRANSERNODE of the **SrvRoom** where the *Entity Destination Type* has been set to continue, the destination node that was set using the *Set Node* step will be used to route the entities.

Properties: GotoSafetyDeposit (SetNode Step Instance)		Properties: GotoExit (SetNode Step Instance)	
Basic Logic		Basic Logic	
Destination Type	Specific	Destination Type	Specific
Node Name	Input@SrvSafetyDeposit	Node Name	Input@SnkExit

**Figure 15.8: Property Settings for the Set Node Property to Send the Entity to the Vault or the Exit**

**Step 4:** Insert the “*After\_Processing*” add-on process trigger for the **SrvSafetyDeposit** that will release the **WkrBanker** once the service is completed, as seen in Figure 15.9.<sup>237</sup>



**Figure 15.9: Releasing the Banker after the Customer is finished at the Deposit Box**

**Step 5:** Save and run the model.

*Question 4:* What is the advantage of defining a property instead of specifying 0.45 directly in the *Decide*?

---

*Question 5:* Do some of the customers flow to the safety deposit vault server?

---

*Question 6:* Does the banker resource wait until the customer is completely done visiting the vault before heading back to see another customer?

---

*Question 7:* If the previous question is true, where does the Banker wait while the customer is visiting?

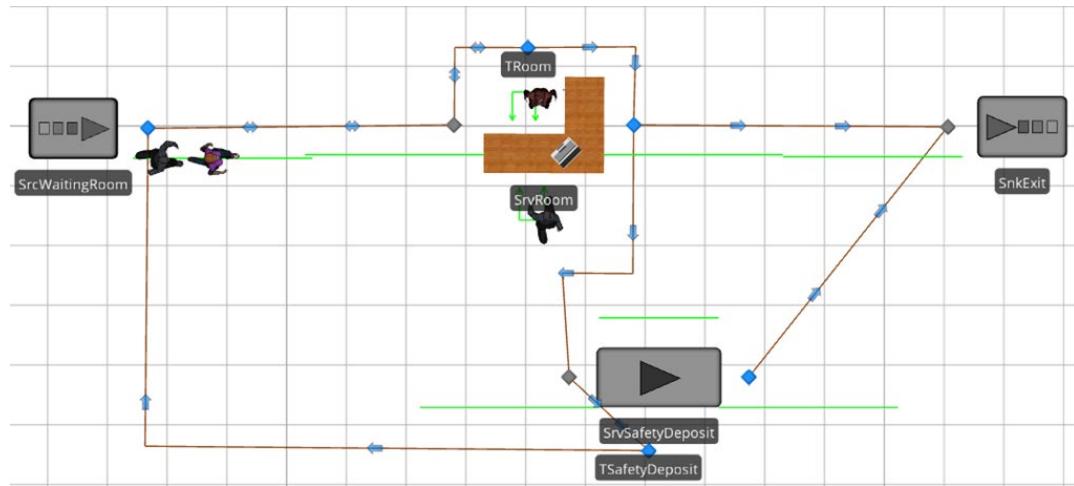
---

**Step 6:** The **WkrBanker** does not move with (i.e., escort) the customer from the room to the safety deposit machine. To facilitate the moving of the **WkrBanker** to the **SrvSafetyDeposit**, paths have to be added to allow the **WkrBanker** to move to the vault room, as seen in Figure 15.10.

- Insert a new TRANSERNODE named **TSafetyDeposit** near the **SrvSafetyDeposit** which will be the node the **WkrBanker** will transfer to for processing the customer at the safety deposit machine.
  - Insert a one-meter path from the **TRoom** transfer node to the output node of the **SrvRoom** so the **WkrBanker** can use the same path to escort the customer to the safety deposit box.
  - Connect the input of the **SrvSafetyDeposit** to the **TSafetyDeposit** transfer node via a one-meter PATH and then connect the **TSafetyDeposit** node back to the output of the **SrvWaitingRoom** via a 15-meter PATH, which allows the **WkrBanker** to travel back to the waiting room to get the next customer.
- 

<sup>236</sup> In the previous chapter, the SeizedResources functions were used in the link weights to route the entities to either the exit or the X-ray machine.

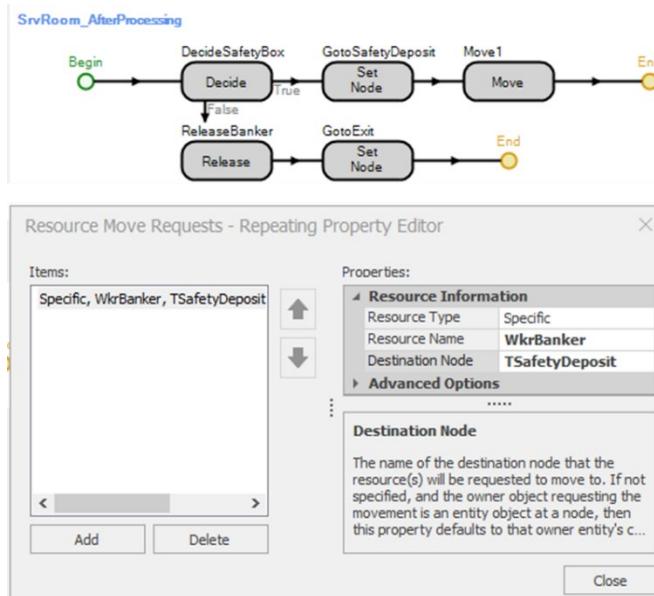
<sup>237</sup> Just copy the *Release* step from the **SrvRoom\_AfterProcessing** to the **SrvSafetyDeposit\_AfterProcessing**.



**Figure 15.10: Model for Transferring Customers and the Banker**

If the customer needs to see his safety deposit box, the **WkrBanker** needs to travel to accompany them to the room. The process logic in the *After\_Processing* add-on trigger in Figure 15.7 moves the entities to either the safety box or the exit based on a probabilistic *Decide* branch using a *Set Node* process step. When this process trigger runs, the TOKEN that is executing the process steps is associated with the entity. Therefore, any steps (i.e., Transferring, Setting Nodes, etc.) are only applied to the entity<sup>238</sup> associated with the TOKEN.

**Step 7:** Insert a *Move* step, which will request a move of one or more moveable resources that have been seized by the MODELENTITY (i.e., the associated object), as seen in Figure 15.11.



**Figure 15.11: Using the *Move* Step to cause the Banker to Move to the Safety Deposit Box**

**Step 8:** Save and run the model, observing what happens with the banker and customer.

**Question 8:** Does the banker now move to the safety deposit box with the customer?

---



---

<sup>238</sup> In some process steps, the action can be applied to the parent object of the Token's associated object (e.g., the Server the logic resides).

**Step 9:** It may appear to move with the customer based on the speed of the animation. However, the banker will actually move to the **TSafetyBox** node first before the customer does. Place a breakpoint on the path connecting the **TRoom** and **Output@SrvRoom** nodes to see this phenomenon. Fast forward the simulation till it stops, and then slow down the animation speed under the “Run” tab.

*Question 9:* Does the banker move first now?

---

**Step 10:** When the customer (i.e., **MODEENTITY**) requests a dynamic resource to be moved using the **Move** step, the token associated with the entity will wait until the movement occurs before proceeding forward (i.e., move itself). Therefore, we would like the banker and customer to move independently. To manipulate the banker, it will be necessary to identify the banker and then influence it. To accomplish that in this model we will employ a **Search** process step. The **Search** process step can search a collection of items and return Tokens associated with the items as specified in Table 15.1 and Table 15.2.

**Table 15.1: The Properties of the Search Process Step**

Property	Description
<i>Collection Type</i>	Type of the collection to search (See Table 15.2 for more information).
<i>Search Type</i>	The type of search to perform (Forward, Backward, MinimizeReturnValue, and MaximizeReturnValue). <sup>239</sup>
<i>Match Condition</i>	Optional match condition that can be used to filter the items in the collection before the search begins. Don’t forget to use the keyword <b>CANDIDATE</b> in the condition.
<i>Search Expression</i>	An expression that is evaluated for each item found by the search. The total sum for this expression is returned in the <b>ReturnValue</b> of the original <b>TOKEN</b> . One could save the particular Resource and Seize the exact same one later.
<i>Starting Index</i>	One based index to start searching. By default it is one for Forward searches while the number in the collection for Backward searches.
<i>Ending Index</i>	One based index to end searching to assist in narrowing the search within collection.
<i>Limit</i>	An expression that specifies the maximum number of items that can be found and returned.
<i>Save Index Found</i>	Specify an optional discrete state variable to save the last item found.

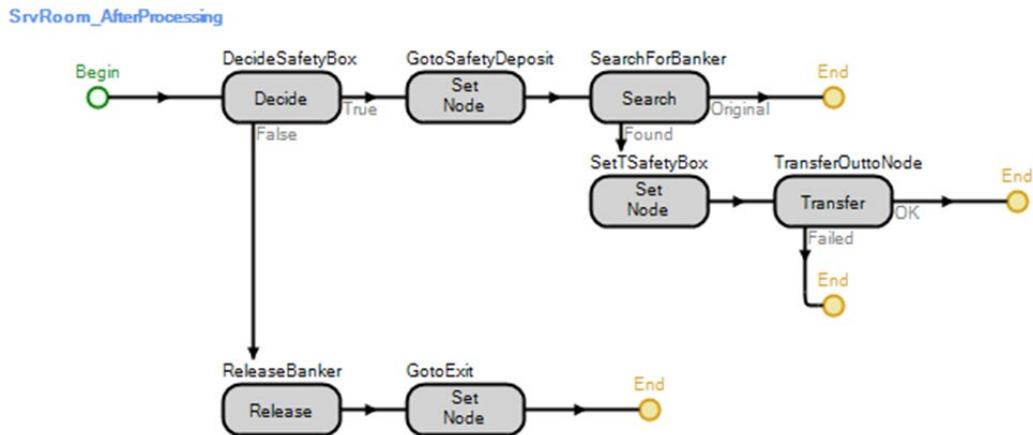
**Table 15.2: Different Types of Collections that Can Be Searched**

Collection Type	Description
<b>ObjectInstance</b>	Search all objects of a particular class (e.g., search all the entities that are currently created)
<b>EntityPopulation</b>	Search a particular entity population (i.e., <b>MODEENTITY</b> , <b>WORKER</b> , or <b>TRANSPORTERS</b> )
<b>ObjectList</b>	Search the objects in a list (e.g., <b>ResourceList</b> , etc.)
<b>NodeList</b>	Search all nodes in a particular list.
<b>TransporterList</b>	Search all transporters in a particular list.
<b>SeizedResources</b>	Search the objects currently seized by the Parent Object, Associated Object or a Specific Object
<b>QueueState</b>	Search the objects currently in one of the queues
<b>TableRows</b>	Search the rows in a Data table
<b>NetworkLinks</b>	Search all links of a particular network.
<b>NodeInboundLinks</b>	Search all the inbound links of a particular node.
<b>NodeOutboundLinks</b>	Search all the out bound links of a particular node.

---

<sup>239</sup> **MinimizeReturnValue** and **MaximizeReturnValue** search values will search the collection and return up to the limit of items specified that minimizes or maximizes the Return Value expression property.

**Step 11:** If the customer needs to be escorted to the safety deposit box, insert a *Search* Step after the *Set Node*, as seen in Figure 15.12. The “Original” branch is associated with the original TOKEN (i.e., customer), while the “Found” branch will be invoked for a Token associated with each item found.



**Figure 15.12: Searching for and Transferring the Banker**

- Set the *Collection Type* property to *SeizedResources* and make sure the *Owner Type* is “AssociatedObject” which will search all the objects that have been seized by the customer.
- Set the *Match Condition* to *Candidate.Object.Is.WkrBanker* to only search for the banker that has been seized.<sup>240</sup> This condition will ensure that only the seized banker is.
- Set the *Search Expression* property to return the “ID” of the *WORKER* seized, which can be used for a further enhancement.

Properties: SearchForBanker (Search Step Instance)	
Basic Logic	
Collection Type	<b>SeizedResources</b>
Owner Type	AssociatedObject
Search Type	<b>ForwardSumExpression</b>
Match Condition	<b>Candidate.Object.Is.WkrBanker</b>
Search Expression	<b>Candidate.Worker.ID</b>

**Figure 15.13: Properties of the Search Process Step**

**Step 12:** Once the **WkrBanker** has been found, we need to set the node to the **TSafetyBox** and then using a *Transfer*<sup>241</sup> process step transfer out of the parking station (i.e., the current stations) and into the **TRoom** node as seen in Figure 15.12 with the properties as in Figure 15.14.<sup>242</sup>

Properties: SetTSafetyBox (SetNode Step Instance)	
Basic Logic	
Destination Type	Specific
Node Name	<b>TSafetyDeposit</b>

Properties: TransferOuttoNode (Transfer Step Instance)	
Basic Logic	
From	<b>CurrentStation</b>
To	Node
Node Name	<b>TRoom</b>

**Figure 15.14: Transfer Process Step Properties to Move Banker to the Vault**

<sup>240</sup> The customer entity has also seized one unit of capacity from the Server and we do not want to return the Server object.

<sup>241</sup> The *Transfer* step can be used to transfer dynamic objects (Agents, Entities, Vehicles, etc.) from Current Node, Current Station, Free Space, or Link to a specific Node, another station, Parents External Node, or outbound link.

<sup>242</sup> If you transfer it directly to the **TSafetyBox** it will beam the banker there like Star Trek and not route it as this will take it out of the station and place in the **TRoom** node which then cause the banker to move to the **TSafetyBox** node.

**Step 13:** Save and run the model.

*Question 10:* Does the banker now move with the customer when it needs to visit the safety deposit box?

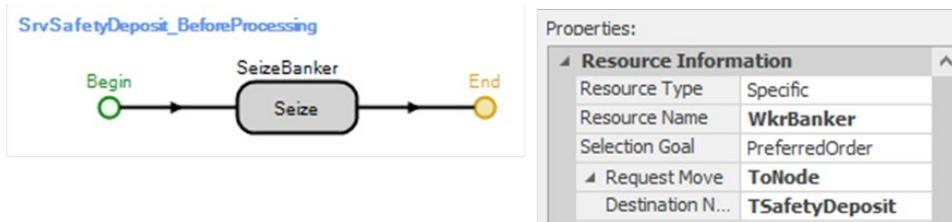
*Question 11:* Does it behave the way you envisioned?

### Part 15.3: Using the Transport Function of the Worker

The difficulty of the previous model, the customer and the banker worker moved independently of one another. Instead, the banker should pick up the customer (i.e., act as a vehicle) and then transport the customer to the safety deposit vault, where the customer would then *Seize* it again, similar to moving the customer to the banker's office. Here, the banker will serve the customer and provide transportation.

**Step 1:** Save the current project.

**Step 2:** For the **SrvSafetyDeposit** SERVER, insert a *Before\_Processing* add-on process trigger with a *Seize* step (or you can use the *Secondary Resources*) to seize the **WkrBanker** once the customer is dropped off. Use the *Seize* step in the *Before\_Processing* that will seize the **WkrBanker** and request a move to the **TSafetyDeposit** node, as seen in Figure 15.15.



**Figure 15.15: Seize the Banker at the Safety Deposit Vault**

**Step 3:** However, only certain entities need to ride on a worker/vehicle (i.e., the ones heading to the safety deposit box) while the other customers will walk to the exit. Therefore, the output node at the **SrvRoom** cannot be specified to “Ride on Transporter” like the output node for the **SOURCE** since this would force all customers to ride. A *Ride* process step can be used to force an “Entity to Ride” on a vehicle. However, the *Ride* process step can only be utilized in a process trigger inside a **TransferNode** (e.g., **Output@SrvRoom**). Therefore, all of the logic used to determine where the entity will proceed after processing must be removed.

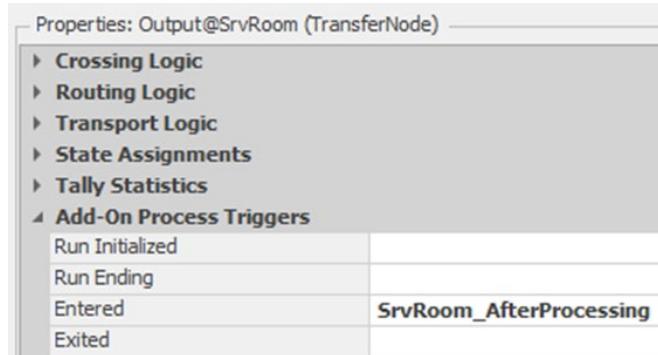
- Remove the *After Processing* add-on process trigger from the **SrvRoom** by choosing “null” from the dropdown<sup>243</sup> or using “reset” on the right-click since this logic will be needed in a different object’s add-on triggers, as seen in Figure 15.16.

Before Processing	<b>SrvRoom_BeforeProcessing</b>
Processing	
After Processing	

**Figure 15.16: Removing an Add-on Process Trigger from an Object**

<sup>243</sup> You can also remove the process trigger by selecting it and then deleting it. Removing the process trigger only removes the process from being invoked and does physically remove it

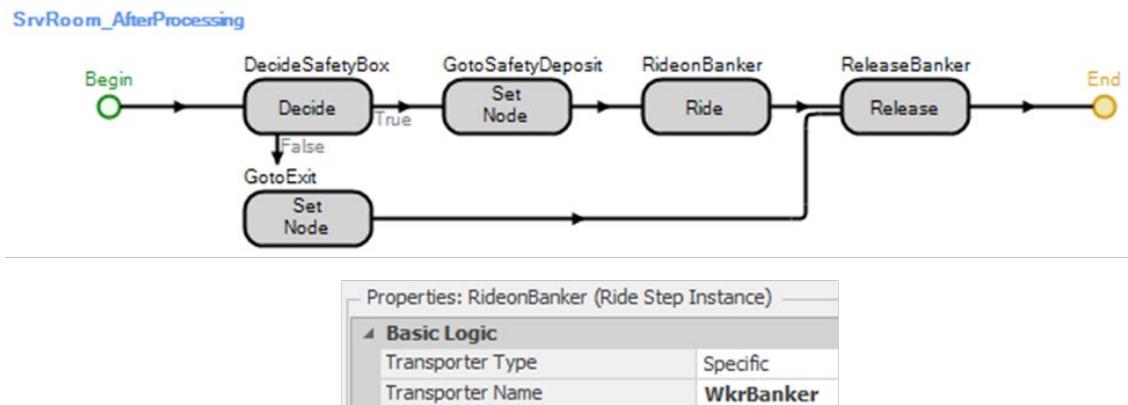
- Select the output TRANSERNODE for the SrvRoom and then choose the SrvRoom\_AfterProcessing<sup>244</sup> process as the “Entered” add-on process trigger, as seen in Figure 15.17. The Entered add-on process will be fired when an entity, worker, or vehicle enters the TRANSERNODE.



**Figure 15.17: Specifying the Entered Add-On Process Trigger**

**Step 4:** If the customer needs to visit the vault, the customer should request a ride and then release the WkrBanker so it can respond to ride requests.

- In the SrvRoom\_AfterProcessing process, delete the *Search* step since it will not be needed.
- Insert a *Ride* step with the *Transporter Type* set to “Specific” and the specific *Transporter Name* specified as a WkrBanker (see Figure 15.18).
- If the entity needs to see their safety deposit box and the customer has requested a ride, the WkrBanker needs to be released so it can pick up the customer in order to transport it to the safety deposit box. To reuse the same *Release* in the “False” branch, drag the end of the branch (End) associated with the *Ride* step and drop it on top of the *Release* step.<sup>245</sup>



**Figure 15.18: Adding the Ride Step and then Releasing the Banker**

**Step 5:** Save and run the model.

**Question 12:** What do you observe when the customer needs to visit the safety box now?

---

<sup>244</sup> The SrvRoom\_Processed is just the name of the process and can be easily changed. You could have created a new process trigger and then copied all the logic into the new process.

<sup>245</sup> You could have copied and pasted another *Release* step after the *Ride* instead

**Step 6:** The banker is stalled at the desk while the customer waits for a ride to the safety deposit box. More specifically, the TOKEN associated with the entity in the process is waiting at the *Ride* step owing to the *Token Wait Action* property (within the *Advanced Options*), which specifies, by default, to “WaitUntilTransferred” (i.e., until the entity has finished being transferred into the vehicle). Now change the *Token Wait Action* property to “NoWait,” as shown in Figure 15.19.

Properties: RideonBanker (Ride Step Instance)	
Basic Logic	
Transporter Type	Specific
Transporter Name	<b>WkrBanker</b>
Reservation Method	ReserveClosest
Selection Goal	PreferredOrder
Advanced Options	
Selection Condition	
Token Wait Action	<b>NoWait</b>

Figure 15.19: Changing the Token Wait Action

**Step 7:** Save and run the model.

*Question 13:* What do you observe when the customer needs to visit the safety deposit box and the **WkrBanker** moves to the output node of the **SrvRoom**?

---

*Question 14:* Does it behave the way you envisioned?

---

**Step 8:** When the **WkrBanker**, which is a **VEHICLE**, reaches the output node of **SrvRoom** to pick up the customer, it actually enters the node, thus running the same logic in Figure 15.18, which tells it to exit or get picked up by a banker. This then creates the system deadlock. Only customers should run the logic.

**Step 9:** Insert another *Decide* step (see Figure 15.20) before the current *Decide* step, which checks to see if the object associated with the Token is a **MODEL ENTITY**. The *Is* operator will check to see if the object is a particular object, which, in this case, *Is .ModelEntity* will return “True” if it is a customer and “False” if it is a Banker (i.e., **WORKER/VEHICLE**), as seen in Figure 15.20.

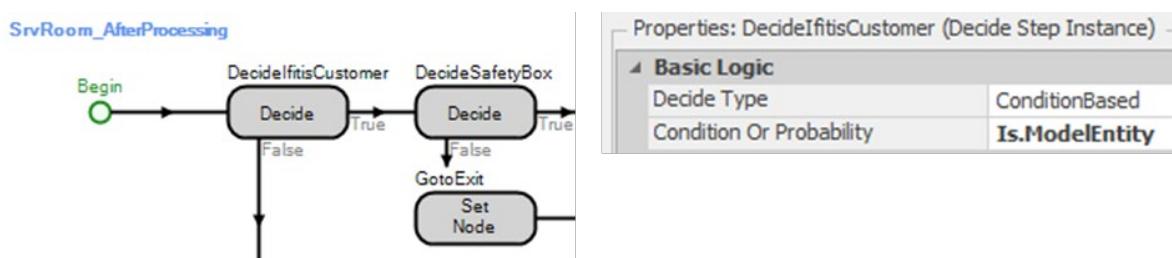


Figure 15.20: Using the “Is” Operator to determine if an Object is an Entity

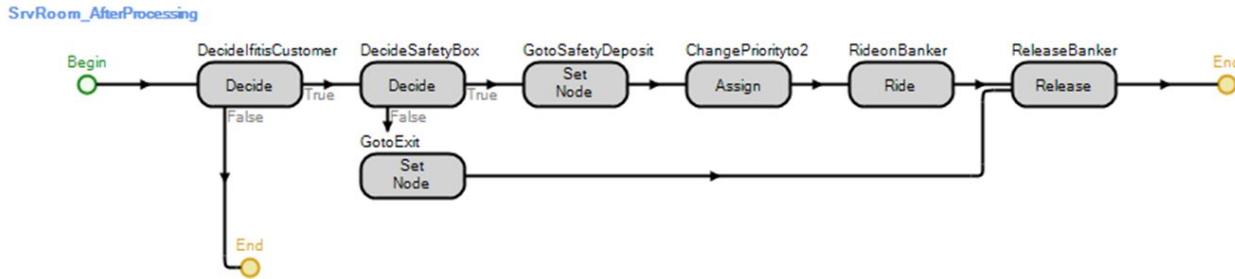
**Step 10:** Save and run the model.

*Question 15:* Does the model behave the way you envisioned?

---

**Step 11:** The problem is the **WkrBanker** responds to ride requests in the order they arrive. Customers going to the safety deposit machine should be given priority.

- Add an *Assign* step that changes the customer's priority (`ModelEntity.Priority`) to two (see Figure 15.21).
- Select **WkrBanker** and then change the *Task Selection Strategy* property under the *Transport Logic* properties to be the “Largest Priority,” as seen in Figure 15.22.<sup>246</sup>



**Figure 15.21: Using the Entered Add-On Trigger to Specify a Ride on Vehicle**

Transport Logic	
Initial Ride Capacity	1
Task Selection Strategy	<b>Largest Priority</b>
▶ Load Time	0.0
▶ Unload Time	0.0
Park to Load/Unload	False
Minimum Dwell Time Type	No Requirement

**Figure 15.22: Changing the Task Selection Strategy of a Vehicle**

**Step 12:** Save and run the model.

**Question 16:** Does the model work as intended now?

## Part 15.4: Resource Reservations

The previous section illustrated a worker's use for service and transportation. However, continued use of this resource during a sequence of needs required a careful set of seizes, rides, and releases. You should note that if someone had requested a seizure of the banker, it would have handled that request before responding to the ride request, regardless of the priority. In these cases, it is easier to use a resource reservation. An entity may “reserve” a resource, worker, or vehicle for continued use at different stages. By reserving it, the entity may use it immediately or later. While being reserved, the resource cannot be used by another entity. A resource may be reserved at a `TRANSFERNODE` or in the “Resource for Processing” in the “Secondary Resources” section of an object or in a process using a *Reserve* step or the *Release* step. Reservations may be implicitly canceled using the *Release* step or explicitly using the *Unreserve* step. Reservations made through the Secondary Resources will be automatically canceled when service is complete. So, each customer arriving at the bank will reserve the banker for use throughout the process until the banker is no longer needed.

**Step 1:** At the `Output@SrcWaitingRoom`, modify the “Transport Logic” to cause the transporter to be reserved as shown in Figure 15.23. Adding this specification causes the banker to be reserved for this customer and unavailable to any other customer for any type of service until the reservation is canceled.<sup>247</sup>

<sup>246</sup> Changing the *Task Selection Strategy* reorders the Rider Pickup Queue of the Vehicle.

<sup>247</sup> The act of cancelling a reservation is called unreserving.

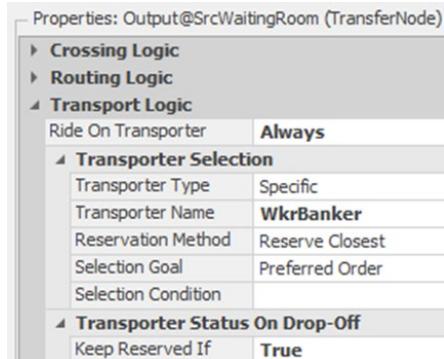


Figure 15.23: Creating a Reservation at Transport

**Step 2:** The customer requests service from the reserved banker at the office. After the banker serves at **SrvRoom**, the customer will either release the banker and exit or keep the banker to be escorted to the safety deposit. Change the **SrvRoom\_AfterProcessing** to have a separate *Release*, as shown in Figure 15.24. The *Assign* step is no longer needed as we will continue reserving the **WkrBanker**.

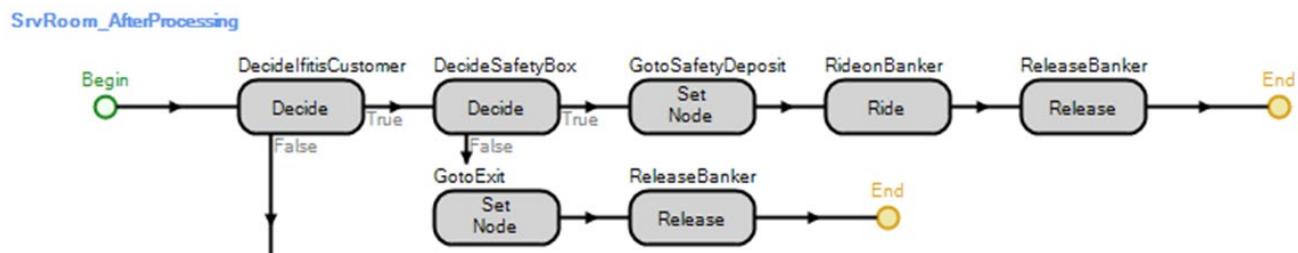


Figure 15.24: Modifying the Process

**Step 3:** When the customer exits, the banker is released. Unless the *Release* step specifies keeping the reservation, the reservation is automatically canceled. However, those customers who go to the safety deposit box keep the reservation in the *Release* step in the process going to the safety deposit box, as shown in Figure 15.25. Note the *Reservation Timeout* expression property allows you to specify logic that will cancel the reservation if the entity still has reserved the object when the time expires.

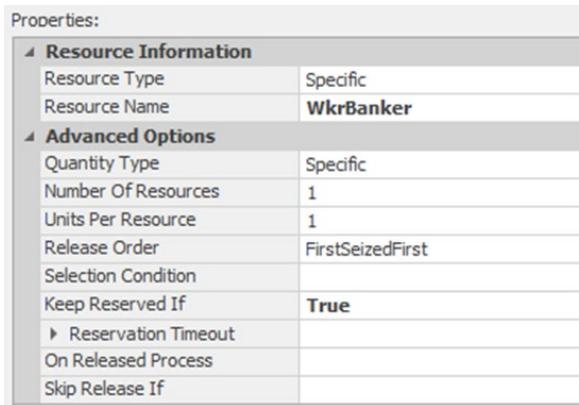


Figure 15.25: Keeping Reservation on Release

**Step 4:** Rerun the model, looking closely at the behavior of the banker.

**Question 17:** Does the model work in a similar fashion as before?

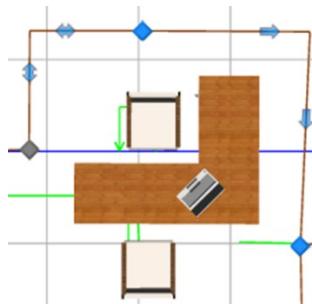
## Part 15.5: Animated Entities

The “animated people” that have been added to SIMIO can enhance the visual appeal of a simulation. Each animated people symbol (e.g., female, male, children, elderly, soldier, and cartoon people) has a list of animation options. You can see the options by right-clicking an animated *ModelEntity/Worker/Vehicle* and selecting “*List Animations of Active Symbol*.” Table 15.3 shows the lists of male and female animated people. You can invoke an animation by its “Index” or its name “Name.” Now, the animation lists are identical across people’s symbol groups, as can be seen in list.

**Table 15.3: Partial List of Animated Symbols for Male and Female Animated People**

Index	Name	Index	Name
1	Stand	12	Walk Carrying Left
2	Stand Shifting Weight	13	Walk Pushing
3	Stand Talking on Phone	14	Run
4	Stand Texting	15	Sit
5	Stand Picking Up	16	Sit Legs Crossed
6	Stand Moving Hands	17	Sit Talking
7	Stand Carrying Front	18	Sit Moving Hands
8	Stand Carrying Side	19	Sit Driving
9	Walk	20	Sit to Lie Down
10	Walk Carrying Front	21	Sleep
11	Walk Carrying Right	22	Dance

**Step 1:** To illustrate the use of animated entities, let’s reconsider the model from the first section of this chapter. Add two chairs at the desk, one for the banker and one for the customer, using the *Drawing → Place Symbol*, as shown in Figure 15.26. You may need to adjust the placement of the chairs, the parking station queue, and the processing station queue as you see the animation perform.



**Figure 15.26: Add Chairs around Desk**

**Step 2:** One of the *MODEL ENTITY* state variables that SIMIO defines in the *Definitions→ States* is “Animation.” This “string” state variable can be used to hold the current entity animation and it can be used to assign a new animated behavior. Specify the “Animation” properties section of the **EntCustomer** as shown in Figure 15.27.

Animation	
Current Symbol Index	ModelEntity.Picture
Random Symbol	True
Current Animation In...	ModelEntity.Animation
Default Animation Ac...	Moving
Link Segment Transiti...	Smooth
Draw Type	Single
Dynamic Label Text	

Figure 15.27: Animation Properties of the Customer

**Step 3:** WORKERS and VEHICLES do not have individual state variables like MODELENTITIES. So, we need to define a new Model DISCRETE STRING STATE variable named **GStringStateWorker** for the banker, which can be changed. So now specify the “Animation” properties of the **WkrBanker** as shown in Figure 15.28.

Animation	
Current Symbol Index	Worker.ResourceState
Random Symbol	False
Current Animation Index	<b>GStaStateWorker</b>
Default Animation Action	MovingAndIdle
Link Segment Transition Type	Smooth
Draw Type	Single
Dynamic Label Text	

Figure 15.28: Animation Properties of the Banker

**Step 4:** The customer should walk to the banker’s desk and then sit in the chair. So, in the *Before Processing* add-on process trigger of the **SrvRoom** SERVER, add the assignment shown in Figure 15.29.

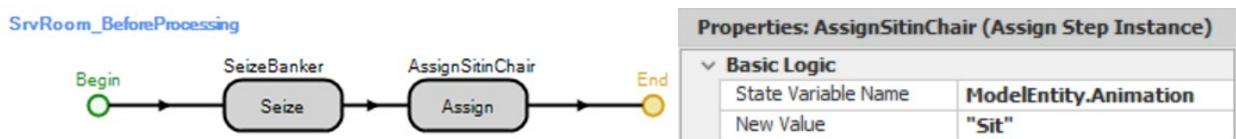


Figure 15.29: Customer Seats at Banker’s Desk

**Step 5:** Then the customer should walk to the exit or the safety deposit room. So, in the *SrvRoom\_AfterProcessing* process, add the assignment shown in Figure 15.30.

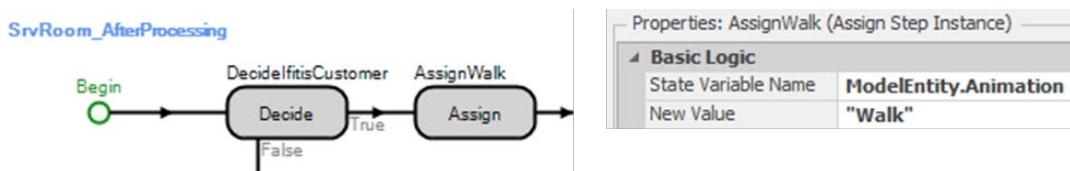


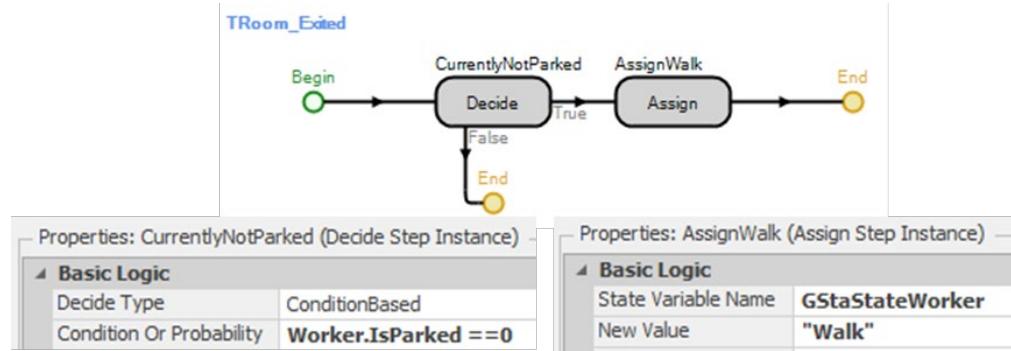
Figure 15.30: Customer Walks to Exit

**Step 6:** The banker needs to sit at the desk. So, in the “Entered” add-on process trigger of the **TRoom** TRANSFERNODE, add the assignment shown in Figure 15.31.



Figure 15.31: Banker Seats

**Step 7:** The banker needs to walk again when they leave the desk to pick up the next customer or take the current customer to the safety deposit box. So, in the *Exited* add-on process trigger, add the *Decide* and *Assign* step, as shown in Figure 15.32. Recall that we specified that the **WrkBANKER** would park when they were seized. When the **WORKER** object enters the parking station, it exits the node and runs the *Exited* add-on process trigger, forcing the banker to walk after they were told to sit. The *IsParked* function returns true once the worker enters the parking station before exiting the node and, therefore will not set the variable to “Walk.”<sup>248</sup>



**Figure 15.32: Banker Walks**

**Step 8:** Save and run the model by looking carefully at the animation<sup>249</sup>.

*Question 18:* Are the animations working as expected?

## Part 15.6: Detailed Service: Tasks and Task Sequences

A service activity may be composed of several tasks rather than a simple processing time. For example, the kitting of an electronic product may employ a sequence of tasks to complete the kit, or an inspection may require a number of tests at the inspection station. Often, the details of these types of operations can be conveniently summarized by their processing time. There are, however, instances when the processing time needs a composition of a set of interrelated tasks. SIMIO, therefore, recognizes two types of processing stations within a SERVER, which are referred to as *Process Types*. Specifically, the alternative specifications are “*Specific Time*” or “*Task Sequence*”. Currently, we have only modeled using the *Specific Time*, which is the default value. When more details about a process need to be included in the model, the model is extended with more objects (i.e., additional servers are inserted). Using a *Task Sequence* process type, details about processing can be incorporated into a set of task sequences within a single SERVER.

Currently, the bank has a loan officer who assists customers with various loans, account information, special deposits, and safety deposit box usage. A single processing time distribution was used to describe that service. In the example, the loan officer escorted customers back to the office for general service. A portion of the customers required a visit to their safety deposit box, which required the loan office to accompany them. In that example, we employed the **WORKER** for service and transportation, breaking up the tasks into two separate servers and flows. We want to reconsider the bank service and employ a different modeling approach to

<sup>248</sup> The worker will enter the node when they leave the parking station as well executing the “*Entered*” add-on process trigger. However, we do not need to add a *Decide* in the entered process because the *Exited* runs after the *Entered* making them walk.

<sup>249</sup> Note that the **TRoom** and **Input@SrvRoom** nodes have two-way traffic and so the animation instructions are invoked in both directions. In this case, that is not a problem, but it could be in other instances.

incorporating details of the office service. More specifically, we want to recognize that the service being provided by the loan officer is composed of a set of “tasks.” Each of the tasks has its own task properties. Some tasks must be performed before others, while some tasks may be performed in parallel (at the same time).

**Step 1:** First, change the “*Process Type*” property of the **SrvRoom** to *Task Sequence* as seen in Figure 15.33.

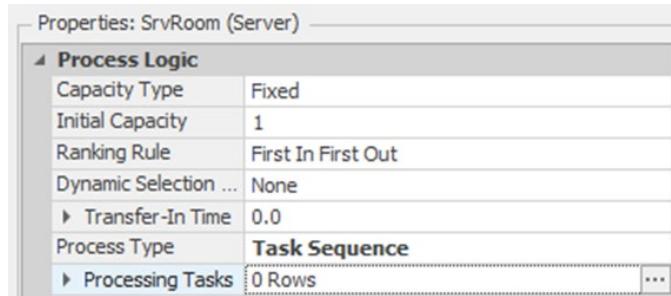


Figure 15.33: Changing Process Type

**Step 2:** This change reveals a *Processing Tasks* property, which is specified through the *Repeating Property Editor* and allows you to list the tasks. Add a task as shown in Figure 15.34. It is important to note that: (1) the *Processing Time* for the task is the same as before, (2) we added the *Name* of the task, and (3) we specified that this task needs the **WkrBanker** and it should be moved to **TRoom** node.

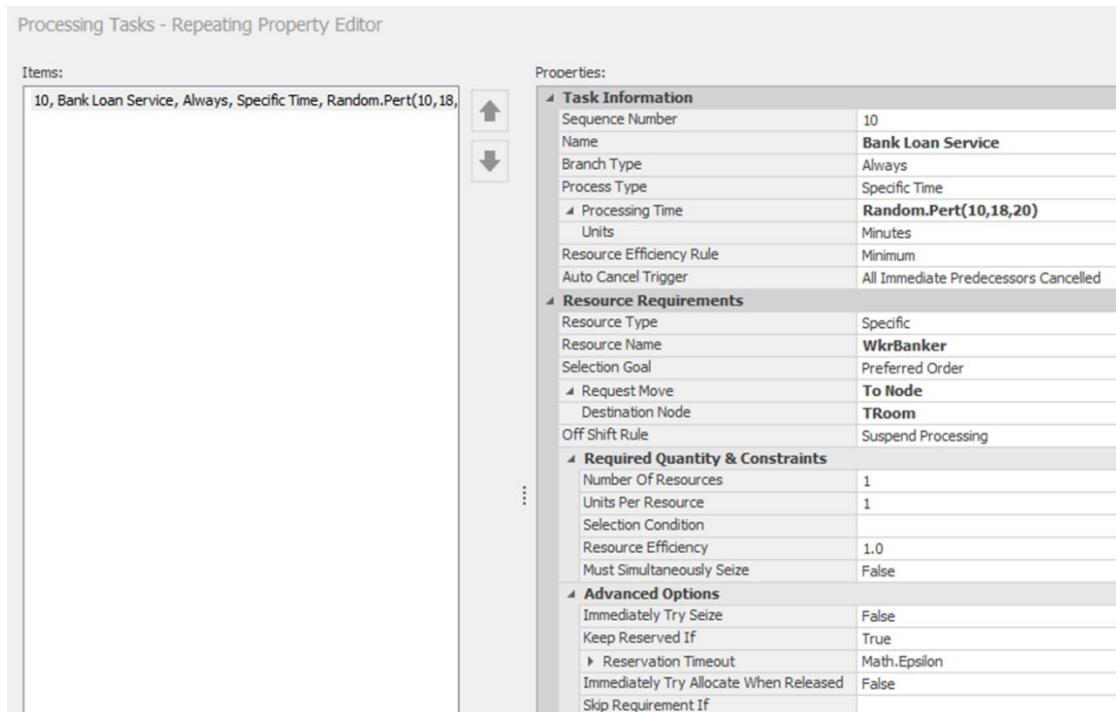


Figure 15.34: Adding a Task

**Step 3:** By default, a *Sequence Number* of 10 is provided. We accepted the *Branch Type* of *Always* and specified the *Process Type* as *Specific Time*. We will modify these specifications in later sections.

**Step 4:** Since we are specifying the use of the **WkrBanker** in the task, we can remove the *Seize* step in the **SrvRoom\_BeforeProcessing** process and the *Release* steps in the **SrvRoom\_AfterProcessing** process. Note that in Figure 15.34, the resource is *Reserved* by default specification of the task. This

specification is for a *Reservation Timeout* of `Math.Epsilon` ensures this resource is held until the end of this time step event, so continued use of the resource is given priority.

*Question 19:* Why is the resource reservation important in this problem? (consider the need to accompany the customer to the safety deposit box)

---

**Step 5:** Save and execute the model.

*Question 20:* Is the model executing as it did previously?

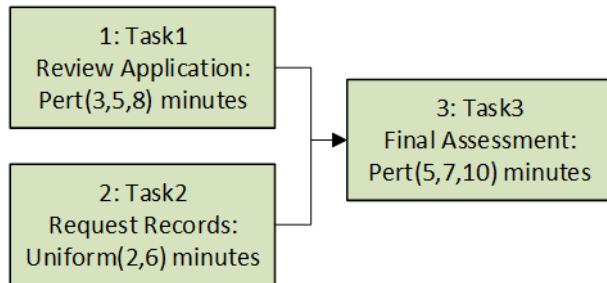
---

*Question 21:* Did the use of a task make it easier to create this model?

---

## Part 15.7: Using Task Sequences

The previous example doesn't illustrate the capability especially since we handled it with processes earlier or could have used secondary resources. A task sequence refers to a network of tasks ordered by precedence. For example, when modeling a patient visit to a doctor's office, you need a nurse to take vitals and information, and then you need a doctor, which could potentially lead to more nurse visits. This would be harder to handle using a single SERVER. Let's consider the banking service example, which has more complications. Suppose the Loan Officer service is really composed of three distinct tasks: (1) Task1: a review of the application, which takes  $\text{Pert}(3,5,8)$  minutes; (2) Task2: a request for records, which only happens in 20% of the cases, and (3) Task3: a final assessment of the loan, which takes  $\text{Pert}(5,7,10)$  minutes. The request for records doesn't require time from the Loan Officer but is an automatic search that takes a  $\text{Uniform}(2,6)$  minutes. In other words, Task 1 and Task 2 are done in parallel, but only Task 1 requires time from the loan officer, and Task 2 may only require time. Task 3 cannot start until both Task 1 and Task 2 are completed. A task sequence diagram is shown in Figure 15.35.



**Figure 15.35: Task Sequence Diagram**

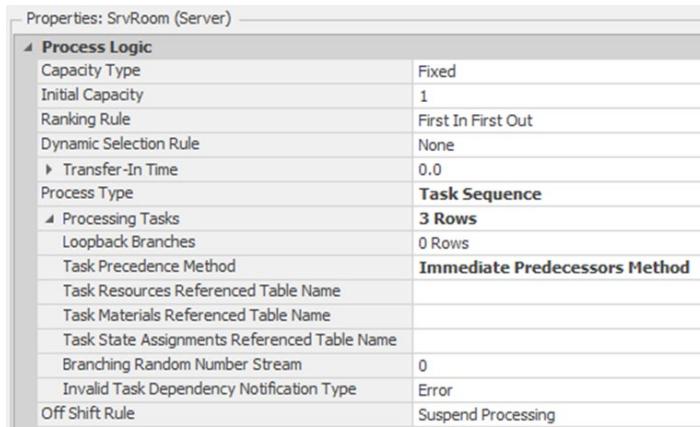
Simio provides three alternatives for specifying task sequences within the SERVER. There are three options to specify the precedence diagram via the *Task Precedence Method*, as given in Table 15.4. In this example, we will use the *Immediate Predecessors Method*, which we find to be the easiest way to specify the precedence.

**Table 15.4: Specifying the Task Precedence Diagram**

Branch Type	Description
<b>Sequence Number</b>	Using a task sequence numbering scheme like XX.YY to determine which tasks precede others (e.g., 10 precedes 20.1 and 20.2 while 20.1 precedes 30.1). See the <b>Commentary</b> section for more information.
<b>Immediate Predecessors Method</b>	Specify all the immediate tasks that need to be completed before the task can start.
<b>Immediate Successors Method</b>	Specify all the successor tasks that need to be started after the task finishes.

**Step 1:** For the **SrvRoom**, we will need to make the following modifications.

- Change the *Task Precedence Method* to the “Immediate Predecessors Method,” as shown in Figure 15.36.



**Figure 15.36 Task Precedence Method**

- Modify the current task to represent the application review process, as seen in Figure 15.37. Set the task *ID Number* to 1. Since this task does not have any predecessors, *Immediate Predecessors* is left blank. Note that the worker is needed for this task and should be moved to the node **TRoom**. When this task is complete, the worker will be released. However, we will keep the worker reserved as it will be needed for the loan assessment task. Since the parallel task, “Request Records,” may take longer than the “Review Application” task, the precise time for the reservation length is unknown. So, we will specify *Keep Reserved If* as “True” and *Reservation Timeout* as “Infinity.”

Task Information	
ID Number	1
Name	ReviewApplication
Branch Type	Always
Process Type	Specific Time
► Processing Time	Random.Pert(3,5,8)
Resource Efficiency Rule	Minimum
Immediate Predecessors	
Auto Cancel Trigger	All Immediate Predecessors Cancelled
Resource Requirements	
Resource Type	Specific
Resource Name	WkrBanker
Selection Goal	Preferred Order
► Request Move	
Destination Node	To Node
Off Shift Rule	TRoom
► Required Quantity & Constraints	Suspend Processing
Advanced Options	
Immediately Try Seize	False
Keep Reserved If	True
► Reservation Timeout	Infinity
Immediately Try Allocate When Released	False
Skip Requirement If	

**Figure 15.37: Review Application Task Information**

- Add a second task named “RequestRecords” with task *ID Number* of 2. Recall the request records task only occurs 20% of the time, as seen in Figure 15.38, and requires no resources in the current scope of the model. Change the *Branch Type* property to “Independent Probabilistic,” as explained in Table 1.5. Again, we will use a property to represent the 20% of time a request is needed. Right-click on the *Condition or Probability* property and specify it to create a new property named **GProbRecordCheck**.

**Table 15.5: Task Branch Type in the Task Sequence Definition**

Branch Type	Description
Always	Branch will be executed all of the time.
Conditional	Branch will conditionally be executed based on the expression independent of any other tasks (i.e., if the expression evaluates to true then the branch is executed otherwise it will be cancelled).
Probabilistic	Branch will be executed probabilistically based on the percentage value stated and is dependent on the other probabilistically branches at the same level. Probabilistic branching is mutual exclusive meaning only one branch or none of the probabilistic branches will be executed. For example, consider a sequence with two tasks (e.g., <i>task A</i> and <i>task B</i> ) each with sequence number 20. If <i>Branch Type</i> is set to “Probabilistic” for both <i>task A</i> and <i>task B</i> with <i>Condition Or Probability</i> values 0.25 and 0.4 respectively. Both <i>task A</i> and <i>task B</i> cannot be executed at the same time. The system will execute <i>task A</i> 25% of the time, <i>task B</i> 40% of the time, and neither task 35% of the time. <sup>250</sup> Note the sum of all the probabilities at the same level cannot exceed 100%.
Independent Probabilistic	Branch will be executed probabilistically independent of other task branch probabilities. For example, if two tasks <i>task X</i> and <i>task Y</i> are specified as “Independent Probabilistic” with probabilities 0.4 and 0.7, respectively, one of the tasks, both of the tasks, or neither of the tasks may be required. Since the tasks are independent, the likelihood of both tasks being required is $(0.4)*(0.7) = 0.28$ .

<sup>250</sup> If you wanted both *task A* and *task B* to be executed independent of each other, then set the *Branch Type* to “Conditional” with the *Condition or Property* value set to  $\text{Random.Uniform}(0,1) < .25$  and  $\text{Random.Uniform}(0,1) < 0.4$  respectively.

Task Information	
ID Number	2
Name	RequestRecords
Branch Type	Independent Probabilistic
Condition Or Probability	GProbRecordCheck
Process Type	Specific Time
Processing Time	Random.Uniform(2,6)
Resource Efficiency Rule	Minimum
Immediate Predecessors	
Auto Cancel Trigger	All Immediate Predecessors Cancelled

Figure 15.38: Request Records Task Information

- For the last task, add a third task, as seen in Figure 15.39. Specify the task *ID Number* of 3, the task *Name* to be “AssessLoan,” and *Processing Time*. Since this task requires both task 1 and task 2 to be completed before it can begin, specify the *Immediate Predecessors* as a comma-separated list of the task *ID Numbers*. Finally, recall that the loan assessment task requires the **WrkBanker** again. However, since the worker is already at the service location, we do not need to make a move request.

Task Information	
ID Number	3
Name	AssessLoan
Branch Type	Always
Process Type	Specific Time
Processing Time	Random.Pert(5,7,10)
Resource Efficiency Rule	Minimum
Immediate Predecessors	1, 2
Auto Cancel Trigger	All Immediate Predecessors Cancelled
Resource Requirements	
Resource Type	Specific
Resource Name	WkrBanker
Selection Goal	Preferred Order
Request Move	None
Off Shift Rule	Suspend Processing

Figure 15.39: Assess Loan Task Information

- Once you close up the tasks, we need to make sure the **GProbRecordCheck** property is set to 0.2. Either right-click on the MODEL in the [Navigation] panel and select *Properties* or select somewhere in the model to access the MODEL properties under the *Controls→General* section, set the value to 0.2.

*Question 22:* Why do we not have to request a move for the banker in this task?

---

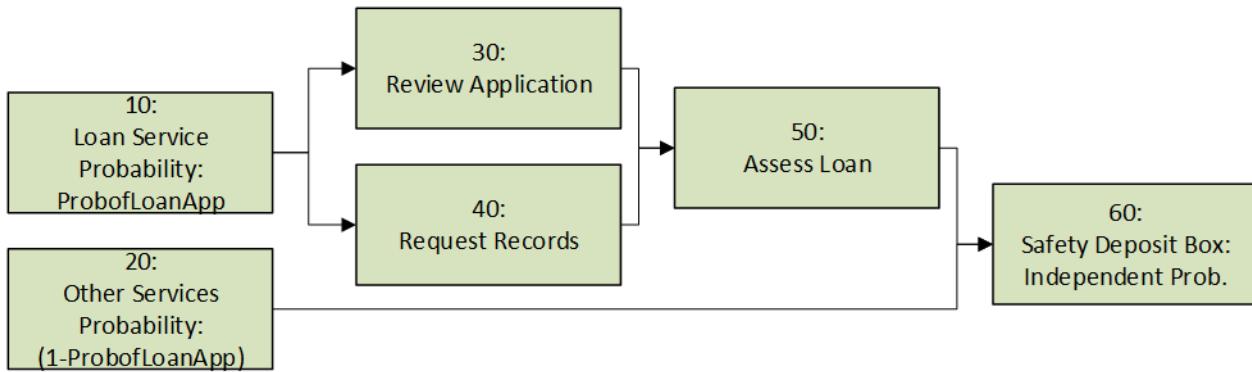
**Step 2:** Run the model.

*Question 23:* Is the model executing as it did previously?

---

## Part 15.8: Some Observations Concerning Tasks

The addition of tasks to SIMIO is rather new, and their potential is still being explored. However, when a service activity needs detailed modeling, the task approach offers some interesting possibilities. For example, let’s assume the banker may have two primary sets of tasks: loan application (75% of the time) and other service visits (25% of the time). Also, for the Assess Loan task, 20% of the customers have issues and take longer to process. For our problem, we would like to model the more complicated set of activities to be performed by the banker at his office, as seen in the task sequence diagram in Figure 15.40.



**Figure 15.40: Single Task Sequence Diagram**

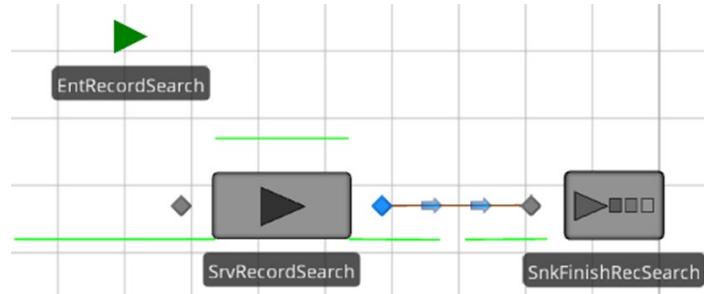
We will again use the “Immediate Predecessors Method” in this task sequence example. Even though for this set of tasks, some can be handled in a similar fashion as the last section (i.e., basic processing time and seizing of resources), we will demonstrate the use of the other process types as described in 2 to show the flexibility by using processes and submodels. We will use processes to model the “Other Services,” “Assess Loan” application, and the “Safety Deposit” tasks, and we will utilize a submodel for the “Request Records” task. For the “Review Application” task we will use the same *Specific Time* process type as well as for task 10, which will be a zero-processing time task needed to complement the probability of performing other services. These process-type options are explained in 2.

**Table 15.6: Process Type Option Explanation**

Process Type	Description
Specific Time	Expression represents the time needed for the task to complete.
Process Name	Specify a process that will be executed when the task starts. The task will finish once the process has finished executing.
SubModel	Allows you to create a different model within the current facility window. When the task starts, an entity of a type that is specified will be created and sent to the starting node of the submodel. You can save a reference to the original entity to pass information to the created entity. The task finishes once the created entity is destroyed by the submodel.

**Step 1:** From the *Definitions→Properties* tab, insert a new “model” *Expression* property named **GProbofLoanApp** with a default value of 0.75 for the probability that a customer needs a loan versus other services. Repeat the process for **GProbLoanAppIssues** for the percentage of customers who have issues with their loan application, setting the default to 0.20.

**Step 2:** As mentioned, we will utilize a submodel for the “Request Records” task, which is done in a different department once the banker submits it. In a different portion of the model, insert a new **MODELENTITY** named **EntRecordSearch**, a **SERVER** named **SrvRecordSearch**, and a **SINK** named **SnkFinishRecSearch**, as seen in Figure 15.41.

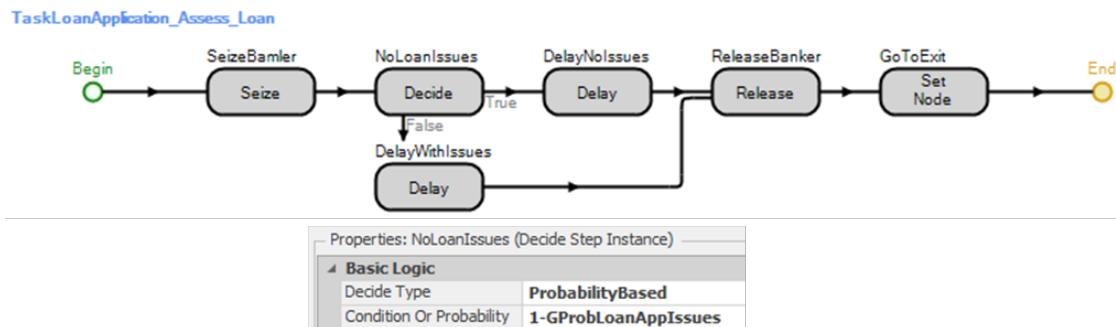


**Figure 15.41: Submodel for Request Records Department**

- Connect the **SrvRecordSearch** to the **SnkFinishRecSearch** via a connector, as we are not concerned with travel time.
- For the SERVER, set the *Processing Time* property to be the same Random.Uniform(2, 6) minutes to represent the time the department needs to process the request for record search.

**Step 3:** For the three tasks (i.e., “Assess Loan,” “Other Services,” and “Safety Deposit Box”), we will create the processes that are associated with each task. Even though we could easily use the *Resource Requirements* to request the **WkrBanker** as before, we will *Seize* and *Release* the banker as part of the process because we can control when they occur for future enhancements.

- Insert a new process named **TaskLoanApplication\_AssessLoan**, as seen in Figure 15.42. The *Seize* step should request the banker to the **TRoom**, as shown previously in Figure 15.5. The *Decide* step should use “ProbabilityBased” with the *Condition Or Probability* property expression sent to 1-GProbLoanAppIssues. Use the same Random.Pert(5, 7, 10) minutes for the loan assessment with loans with no issues and Random.Pert(10, 12, 15) minutes for the assessment of loans with issues. Then, release the banker, making sure to reserve the resource as in Figure 15.25, and then set the destination node of the customer to **Input@SnkExit** so this customer will be routed to the exit.



**Figure 15.42: Process for the Assess Loan Task**

- For the other services task, insert a process named **Task\_OtherServices**. Copy all the steps from the previous process, changing the delay time to Random.Pert(4, 6, 8) minutes.



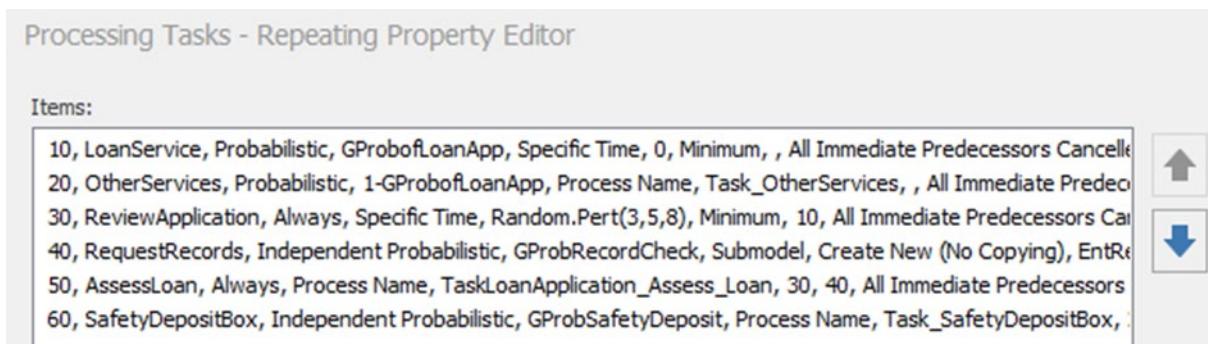
**Figure 15.43: Process for Other Services Performed by Banker**

- Finally, for the task associated with the safety deposit, copy the **Task\_OtherServices** process and rename it to **Task\_SafetyDepositBox**, changing the delay associated with getting the key for the safety deposit box to Random.Pert(2, 4, 6) minutes.



**Figure 15.44: Process for Safety Deposit Box**

**Step 4:** Since we have defined the task submodel and the task processes, we now need to implement the task sequence based on Figure 15.40, as seen in Figure 15.45. We will need to modify/delete the three current task sequences.



**Figure 15.45: Implementation of Task Sequence**

- From Figure 15.40, customers can take two potential sequence paths (i.e., loan application or other services). A *Branch Type* of Probabilistic will be used to select one of these two paths. Therefore, we need to insert a new task (10) named “LoanService” that takes zero time and has a *Condition Or Probability* value equal to **GProbOfLoanApp** that allows the selection of that sequence path.

Task Information	
ID Number	<b>10</b>
Name	<b>LoanService</b>
Branch Type	<b>Probabilistic</b>
Condition Or Probability	<b>GProbOfLoanApp</b>
Process Type	<b>Specific Time</b>
Processing Time	<b>0</b>
Units	<b>Minutes</b>
Resource Efficiency Rule	<b>Minimum</b>
Immediate Predecessors	

**Figure 15.46: Task Select Loan Customer Path**

- Insert a new task (20) named “OtherServices” with 1-GProbOfLoanApp probability to ensure the customer selects either the 10 or 20 sequence task paths. Change the *Process Type* to “ProcessName” and set the *Process Name* property to “Task\_OtherServices.”

Task Information	
ID Number	20
Name	OtherServices
Branch Type	Probabilistic
Condition Or Probability	1-GProbofLoanApp
Process Type	Process Name
Process Name	Task_OtherServices
Immediate Predecessors	

Figure 15.47: Task Associated with Other Services

- Modify the “Review Application” task as shown in Figure 15.48 by changing the sequence number to 30 and specifying the *Immediate Predecessors* as 10. The task will be executed only if the predecessor task is executed owing to the Auto Cancel Trigger property.

Task Information	
ID Number	30
Name	ReviewApplication
Branch Type	Always
Process Type	Specific Time
Processing Time	Random.Pert(3,5,8)
Resource Efficiency Rule	Minimum
Immediate Predecessors	10
Auto Cancel Trigger	All Immediate Predecessors Cancelled
Resource Requirements	
Resource Type	Specific
Resource Name	WkrBanker
Selection Goal	Preferred Order
Request Move	To Node
Destination Node	TRoom
Off Shift Rule	Suspend Processing
Required Quantity & Constraints	
Advanced Options	
Immediately Try Seize	False
Keep Reserved If	True
Reservation Timeout	Infinity

Figure 15.48: Task Associated with Review Application

- Modify the “Request Records” task to execute the submodel instead of using simple processing time as seen in Figure 15.49. Also, change the task number to 40 and specify the *Immediate Predecessors* as 10.

Task Information	
ID Number	40
Name	RequestRecords
Branch Type	Independent Probabilistic
Condition Or Probability	GProbRecordCheck
Process Type	Submodel
Submodel Entity Creation Method	Create New (No Copying)
Submodel Entity Type	EntRecordSearch
Submodel Starting Node	Input@SrvRecordSearch
Save Original Entity Reference	
Immediate Predecessors	10
Auto Cancel Trigger	All Immediate Predecessors Cancelled

Figure 15.49: Modification of the Request Records Task to Execute Submodel

- Modify the “Assess Loan” task by changing the task number to 50 and specifying the *Immediate Predecessors* as a comma-separated list of “30, 40,” as shown in Figure 15.50. Change the *Process Type* to “Process Name” and specify the **TaskLoanApplication\_AssessLoan** process. Also, make sure to remove the *Resource Requirements* of the **WkrBanker**.

Task Information	
ID Number	50
Name	AssessLoan
Branch Type	Always
Process Type	Process Name
Process Name	TaskLoanApplication_Assess_Loan
Immediate Predecessors	30, 40
Auto Cancel Trigger	All Immediate Predecessors Cancelled

Figure 15.50: Modification of the Assess Loan Task to Execute Process

- Finally, insert a new task sequence with a sequence number equal to 60 and the *Immediate Predecessors* as a comma-separated list of “20, 50,” as shown in Figure 15.51. with and use the **Task\_SafetyDepositBox** as the process. Also, make sure the *Branch Type* is “Probabilistic” with the *Condition Or Probability* set to **GProbSafetyDeposit** because only a percentage of customers perform this task.

Task Information	
ID Number	60
Name	SafetyDepositBox
Branch Type	Independent Probabilistic
Condition Or Probability	↳ GProbSafetyDeposit
Process Type	Process Name
Process Name	Task_SafetyDepositBox
Immediate Predecessors	20, 50
Auto Cancel Trigger	All Immediate Predecessors Cancelled

Figure 15.51: Task Sequence for Safety Deposit Box.

**Step 5:** Modify the **SrvRoom\_AfterProcessing** as seen in Figure 15.52. Remove the *Set Nodes* because those are done in the task sequences. Modify the second *Decide* step based on the condition `ModelEntity.DestinationNode == Input@SrvSafetyDeposit`.

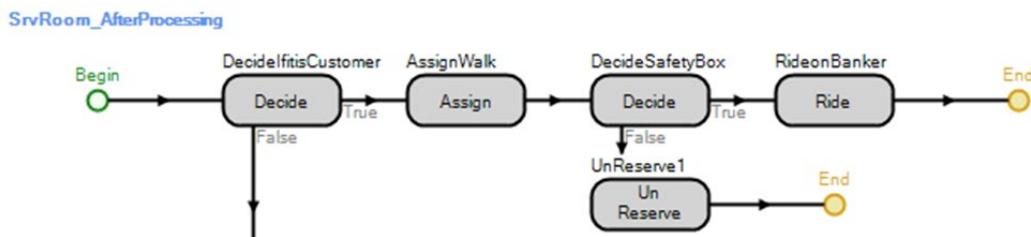


Figure 15.52: SrvRoom\_AfterProcessing

*Question 24:* Why could we not put the *Ride* step in the task sequences?

---

**Step 6:** Save and run the model. You might want to place breakpoints on the **SrvRecordSearch**.

*Question 25:* Does it run as you expected?

---

*Question 26:* Has the use of tasks and task sequences enhanced the modeling of the banker’s service?

---

## Part 15.9: Commentary

- Using the vehicle or the worker for both transportation and service causes the model to become very expressive of the real system.
- Resource reservations make continued use of a resource convenient.
- Tasks and task sequences promote a detailed view of service activity and can incorporate a wide range of behaviors.
- The animated people show promise to greatly enhance the visual appeal of the animation.
- Tasks have two add-on process triggers (i.e., *Starting Task* and *Finished Task*). These processes can be used to record when tasks start and when they end to help calculate critical paths, earliest finish, latest start, etc.
- The general sequence number formation is X.Y.Y.Y, where X and Y are whole numbers. One only needs to define the X value (e.g., 10 and 20), which represents the root number of the sequence and helps to determine the precedence of the task. The Y's are optional integer suffixes separated by periods that are added to the root sequence to define subsequences (e.g., 10.1 and 20.2). Table 15.7 discusses the rules associated with how the task sequence numbers to define the precedence of tasks.

**Table 15.7: Rules of Precedence Associated with Task Sequence Numbers**

Rule	Description
Rule 1	If the task sequence number has only a root number (e.g., 20), all tasks with lower task root numbers, regardless of suffixes, must precede that task (e.g., 10, 10.1, 10.2, 10.1.1 all precede tasks with sequence number 20).
Rule 2	If a <i>task A</i> has a sequence number consisting of the required root number and one or more suffixes (e.g., 20.1, 20.2, or 20.1.1), then tasks with sequence root numbers that are less must precede <i>task A</i> provided they either have no suffixes (i.e., 10 would precede 20.1 or 20.1.1) or match all the suffixes (e.g., 10.1 precedes 20.1, 10.2 precedes 20.2, or 10.1.1 for 20.1.1).
Rule 3	If a <i>task A</i> has a sequence number that has a root number followed by at least one suffix (e.g., 10.1 and 10.2 or 10.1.1 or 10.1.2), then all tasks with the same root number without any suffixes must precede <i>task A</i> (e.g., 10 would precede both 10.1 and 10.2). However, a task with sequence 10.1 does not precede 10.1.1 or 10.1.2, as these will all be considered starting at the same time.
Rule 4	Note one or more tasks may have the same root sequence number with or without suffixes. For example, <i>task A</i> could have sequence number 20 and <i>task B</i> could have sequence number 20, which means these two tasks will start in parallel once the preceding tasks have finished. Starting at the same time is true for tasks with the same root sequence numbers followed by any number of suffixes (e.g., 10.1, 10.1, 10.1.1, or 10.1.2 sequence numbers all start at the same time).

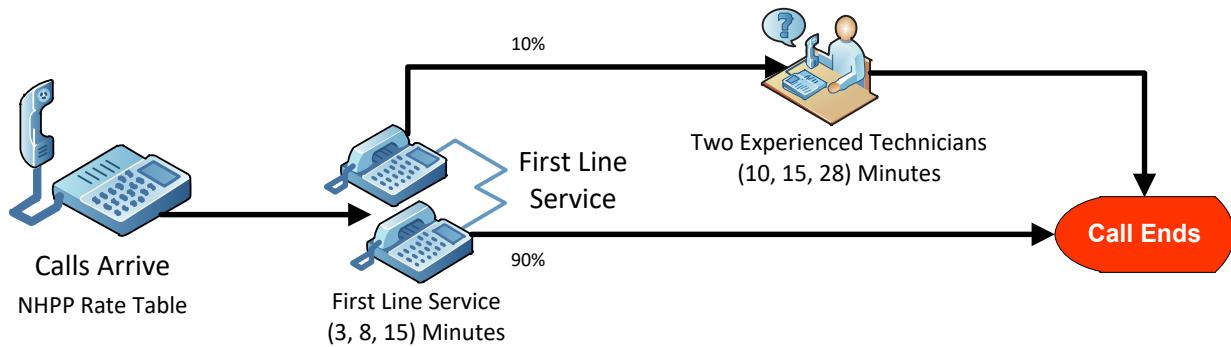
# Chapter 16

## Modeling of Call Centers

---

Previous chapters have extensively used workschedules, non-homogenous rate tables, and routing. This chapter will reinforce those concepts while demonstrating how to handle two of the most common behaviors of people in queues: “balking” and “reneging.”

A call center receives support calls from customers who purchase kitchen appliances. The call center is open seven days a week from 8:00 am to 7:00 pm (i.e., 11 hours). It is, however, staffed from 8:00 am to 8:00 pm, which leaves an hour to finish calls that are in process at 7 pm. There are two lines for technician service. Customers are routed to the technicians (see Figure 16.1) depending on whether or not the first line can answer the customer’s questions.



**Figure 16.1: Model of a Simple Call Center**

Information about the arrival rate of calls during the day is available. Table 16.1 (taken from historical data) displays how the call rates change from hour to hour during the 12-hour day. Note that no calls are taken during the last hour.

**Table 16.1: Hourly Rate of Support Calls to the Center**

Time Interval	Hourly Rate
8 am – 9 am	35
9 am – 10 am	55
10 am – 11 am	70
11 am -12 pm	85
12 pm – 1 pm	95
1 pm – 2 pm	110
2 pm – 3 pm	90
3 pm – 4 pm	88
4 pm – 5 pm	60
5 pm – 6 pm	50
6 pm – 7 pm	30
7 pm – 8 pm	0

Calls are serviced in the order they are received at the call center, and ten support technicians answer the calls. Apparently, the time it takes to answer a call generally takes 8 minutes to answer a support question, with a minimum of 3 minutes and a maximum of 15 minutes. The first-line service technicians can completely answer ninety percent of the calls. So, after working with the customer on the original question, ten percent of the calls must be routed to more experienced technicians, of which there are two technicians. The second-line technicians end up taking about 15 minutes on the call, with a minimum of 10 and a maximum of 28 minutes.

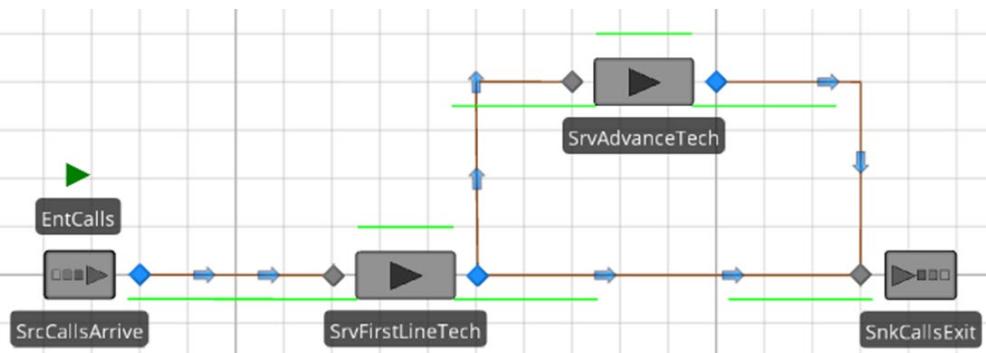
Since the call center uses telephonic equipment, the calls are transferred automatically to the service centers. And, to the best of our knowledge, once a call comes in, the customer stays on the line until it is processed (i.e., currently, there are no call abandonments). When we were able to talk with people at the call center, we found that staffing was not constant throughout the 12-hour day, but there were a number of different schedules for the workers, including part-time and full-time. In fact, the work schedule is described in the following table.

**Table 16.2: Number of First Line Staff per Time Period**

Time Period	Number of First Line Technicians
8 am – 9 am	5
9 am – 10 am	5
10 am – 11 am	8
11 am -12 pm	10
12 pm – 1 pm	10
1 pm – 2 pm	16
2 pm – 3 pm	16
3 pm – 4 pm	12
4 pm – 5 pm	8
5 pm – 6 pm	8
6 pm – 7 pm	8
7 pm – 8 pm	2

### Part 16.1: Building the Simple Model

**Step 1:** Create a new model with one SOURCE named **SrcCallsArrive**, a SINK named **SnkCallsExit**, and two SERVERS named **SrvFirstLineTech** and **SrvAdvancedTech**, as well as add a MODELENTITY named **EntCalls**.



**Figure 16.2: SIMIO Model of Simple Call Center**

**Step 2:** Connect all of the objects via CONNECTORS since these are electronically switched calls. To handle the case that 10% of the calls have to be routed to the advanced technical support line, specify a selection weight of 10 for the connector linking **SrvFirstLineTech** to **SrvAdvanceTech** and a selection weight of 90 for the connector between the **SrvFirstLineTech** and **SnkCallsExit**.

**Step 3:** From the *Data→Rate Tables* section, create a new RATE TABLE named **RateCallTable** with 12 intervals and a one-hour interval size.<sup>251</sup>

Rate Tables			
RateCallTable			
	Starting Offset	Ending Offset	Rate (events per hour)
▶	Day 1, 00:00:00	Day 1, 01:00:00	35
	Day 1, 01:00:00	Day 1, 02:00:00	55
	Day 1, 02:00:00	Day 1, 03:00:00	70
	Day 1, 03:00:00	Day 1, 04:00:00	85
	Day 1, 04:00:00	Day 1, 05:00:00	95
	Day 1, 05:00:00	Day 1, 06:00:00	110
	Day 1, 06:00:00	Day 1, 07:00:00	90
	Day 1, 07:00:00	Day 1, 08:00:00	88
	Day 1, 08:00:00	Day 1, 09:00:00	60
	Day 1, 09:00:00	Day 1, 10:00:00	50
	Day 1, 10:00:00	Day 1, 11:00:00	30
	Day 1, 11:00:00	Day 1, 12:00:00	0

Figure 16.3: Arrival Rate Table for the Call Center

**Step 4:** Return to the model and change the *Arrival Mode* property of the **SrcCallsArrive** source to “Time Varying Arrival Rate” and specify the **RateCallTable** as the *Rate Table* property.

Properties: SrcCallsArrive (Source)	
Entity Arrival Logic	
Entity Type	EntCalls
Arrival Mode	Time Varying Arrival Rate
Rate Table	RateCallTable

Figure 16.4: Specifying the *Arrival Mode* of the Call Arrival SOURCE

**Step 5:** Select the **SrvAdvanceTech** server, change the *Initial Capacity* property to two, and set the *Processing Time* property to `Random.Pert(10,15,28)`, which will simulate two advanced technicians.

**Step 6:** Next, the work schedule of the first-line technicians needs to be created. Recall that SIMIO already defines a default work schedule and day pattern for you. Select the “*StandardDay*” pattern from the “*Day Patterns*” tab under the *Data→Schedules* section and modify it to match the data in Figure 16.5.<sup>252</sup> Let us use the **WORKSCHEDULE** on the left as it specifies every hour separately versus the one on the right, which

<sup>251</sup> We could have modeled the whole 24 hours with the other intervals values being set at zero. However, we changed the run length of our simulation to only be 12 hours.

<sup>252</sup> We find it easier to set the *Start Time* and then the *duration* which will automatically set the *End Time*.

groups some time periods into one for the same values. We will need each time period in a subsequent section.

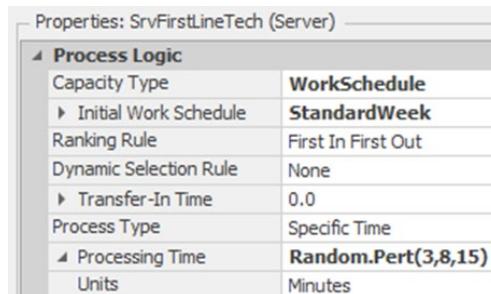
The screenshot shows two windows side-by-side. On the left is the 'Day Patterns' window, which contains a table for 'StandardDay'. This table has a header row 'Work Periods' with columns 'Start Time', 'Duration', 'End Time', 'Value', and 'Cost Multiplier'. Below this are 16 rows representing hourly intervals from 8:00 AM to 7:00 PM. The 'Value' column shows values like 5, 8, 10, 16, etc., and the 'Cost Multiplier' column shows values like 1, 1, 1, 1, etc. On the right is the 'AlternativePattern' window, which also contains a table for 'Work Periods'. This table has columns 'Start Time', 'Duration', 'End Time', 'Value', and 'Cost Multiplier'. It lists time intervals from 8:00 AM to 7:00 PM with corresponding values and multipliers.

Day Patterns					
Name	Description				
StandardDay	Standard 8-5 Work Day				
Work Periods					
Start Time	Duration	End Time	Value	Cost Multiplier	
8:00 AM	1 hour	9:00 AM	5	1	
9:00 AM	1 hour	10:00 AM	5	1	
10:00 AM	1 hour	11:00 AM	8	1	
11:00 PM	1 hour	12:00 AM	10	1	
12:00 PM	1 hour	1:00 PM	10	1	
1:00 PM	1 hour	2:00 PM	16	1	
2:00 PM	1 hour	3:00 PM	16	1	
3:00 PM	1 hour	4:00 PM	12	1	
4:00 PM	1 hour	5:00 PM	8	1	
5:00 PM	1 hour	6:00 PM	8	1	
6:00 PM	1 hour	7:00 PM	8	1	
7:00 PM	1 hour	8:00 PM	2	1	

AlternativePattern					
Work Periods					
Start Time	Duration	End Time	Value	Cost Multiplier	
8:00 AM	2 hours	10:00 AM	5	1	
10:00 AM	1 hour	11:00 AM	8	1	
11:00 AM	2 hours	1:00 PM	10	1	
1:00 PM	2 hours	3:00 PM	16	1	
3:00 PM	1 hour	4:00 PM	12	1	
4:00 PM	3 hours	7:00 PM	8	1	
7:00 PM	1 hour	8:00 PM	2	1	

**Figure 16.5: Work Schedule for the First Line Technicians.**

**Step 7:** Return to the **Model**, select the **SrvFirstLineTech**, and specify the new work schedule and processing time, as seen in Figure 16.6.



**Figure 16.6: Setting the Properties of the First Line Technician Server**

**Step 8:** Even though we do not allow any more calls to come after 7:00 pm, how many people, on average, will be in the queues at 8:00 pm when there are no more technicians to handle these calls? An **OUTPUT STATISTIC** can perform this calculation at the end of the simulation run. From the *Definitions→Elements*, insert a new **OUTPUT STATISTIC** named **OutStatNumberofCallsLeft** with the following Expression property: `EntCalls.Population.NumberInSystem`.

**Step 9:** In the *Run Setup*, change the *Starting Time* to 8:00 am, and the *Ending Type* should be 12 Hours. Save and run the model.

**Question 1:** What is the average customer time in the system, and what is the number of calls still at the end of the simulation?

---

**Question 2:** What is the average utilization of the first line and the experienced technicians?

---

*Question 3:* What is the average holding time in minutes for both of the two server's input buffers?

---

*Question 4:* What is the average number of calls waiting in the two server's input buffers?

---

**Step 10:** Next, insert a new EXPERIMENT and run 20 replications of the model.

*Question 5:* What is the average customer time in the system and the number of calls still in the system at the end of the simulation?

---

*Question 6:* What is the average utilization of the first line and the experienced technicians?

---

*Question 7:* What is the average holding time in minutes for each of the two server's input buffer?

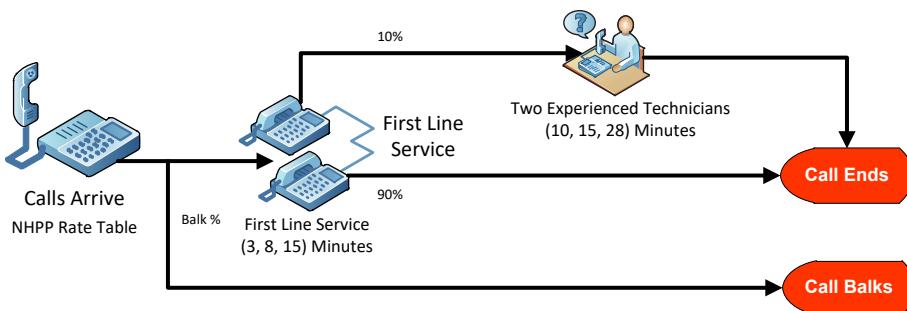
---

*Question 8:* What is the average number of calls waiting in the two server's input buffer?

---

## Part 16.2: Balking

Now, you observe that people who may encounter waiting for a service sometimes abandon that service. In queuing theory, this is called “balking.” Effectively, a person who observes the queue as “too long” simply will not join. In our case, the wait time is proportional to the number in the queue, so we will model the waiting time by using the number (or contents) in the queue. If 20 or fewer people are waiting, only 10 % will balk, while 25% will balk when 21 to 30 people are waiting. And finally, if more than 30 people are waiting, the chance of balking is 50%.



**Figure 16.7: Call Center with Balking**

**Step 1:** Insert a new SINK named **SnkBalk** below **SnkCallsExit** and connect the **SrcCallsArrive** to the new SINK via a CONNECTOR.

**Step 2:** A LOOKUP TABLE offers a great modeling mechanism for predicting the changing balking percentage based on the input queue of the first-line service.<sup>253</sup> From the *Data* tab, create a new LOOKUP TABLE named **FuncLookupBalkPercent** with the appropriate values, as seen in Figure 16.8.

Views		Name
	Tables	LookupBalkPercent
	Data Connectors	
	Lookup Tables	
		X Y
		0 0
		10 0
		11 0.1
		20 0.1
		21 0.25
		30 0.25
		31 0.5
		200 0.5

Figure 16.8: Balking Percentage Lookup Table<sup>254</sup>

**Step 3:** Since the expression is very long, we will set up a function to make it easier and limit errors from occurring. From the *Definitions* tab, insert a new FUNCTION NAMED **FuncBalkPercent** from the *Edit* section. Set the expression to return a value from the Lookup Table based on the number waiting at the *SrvFirstLineTech* input buffer (i.e., `LookupBalkPercent[SrvFirstLineTech.InputBuffer.Contents.NumberWaiting]`) as shown in Figure 16.9.

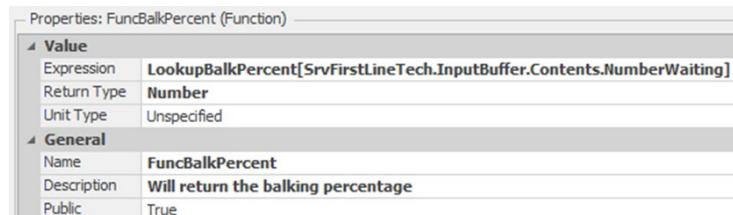


Figure 16.9: Utilizing a Function

**Step 4:** For the connector linking the SOURCE to the SERVER (**SrvFirstLineTech**), set the *Selection Weight* property to `1 - FuncBalkPercent` and the *Selection Weight* property of the link connecting the source to the balking sink to `FuncBalkPercent`.

**Step 5:** Save and rerun the experiment.

**Question 9:** What is the average customer time in the system and the number of calls still in the system?

---

**Question 10:** What is the average utilization of the first line and the experienced technicians?

---



---

<sup>253</sup> There are many ways to model the balking procedure. One could use the balking parameters under the *Input Buffer Logic* in the SERVER. Or, one could add three TRANSFERNODES with the first transfer based on the number in the queue by setting the link weights (i.e., `<= 10`, `> 10 && <= 20`, `> 20 && <= 30`, and `> 30`). Then set the appropriate link weights based on the percentages from the TRANSFERNODES to the **SrvFirstLineTech** or the **SnkBalk**.

<sup>254</sup> It is not strictly necessary to include the values for 0 and for 200, since the table assumes the last value if the lookup is outside the range of the table – a value less than or equal to 10 will be 0 and values equal to or greater than 31 will be 0.5.

*Question 11:* What is the average holding time in minutes for both of the two server's input buffers?

---

*Question 12:* What is the average number of calls waiting in the two server's input buffer?

---

*Question 13:* How many calls, on average, balked during the 12-hour period, and do you feel the current system is handling enough of the calls?

---

### Part 16.3: Modeling Reneging of Customer Calls

Now, we allow customers to renege (i.e., leave) from the queue for standard calls when they have waited for a period of time. In other words, we will allow the abandonment of calls to occur. More specifically, every customer who enters the call queue will wait for at least ten minutes. However, after waiting in the queue for ten minutes, the customer examines its place in the queue (the telephonic system informs them of this). The customer reneges and leaves the system if they are not within the first five places from the front of the queue. If they are within the first five places of the front, then the customer will extend their stay in the queue for another four minutes in hopes of getting served. If they are still in line after the extended stay, they will also renege.

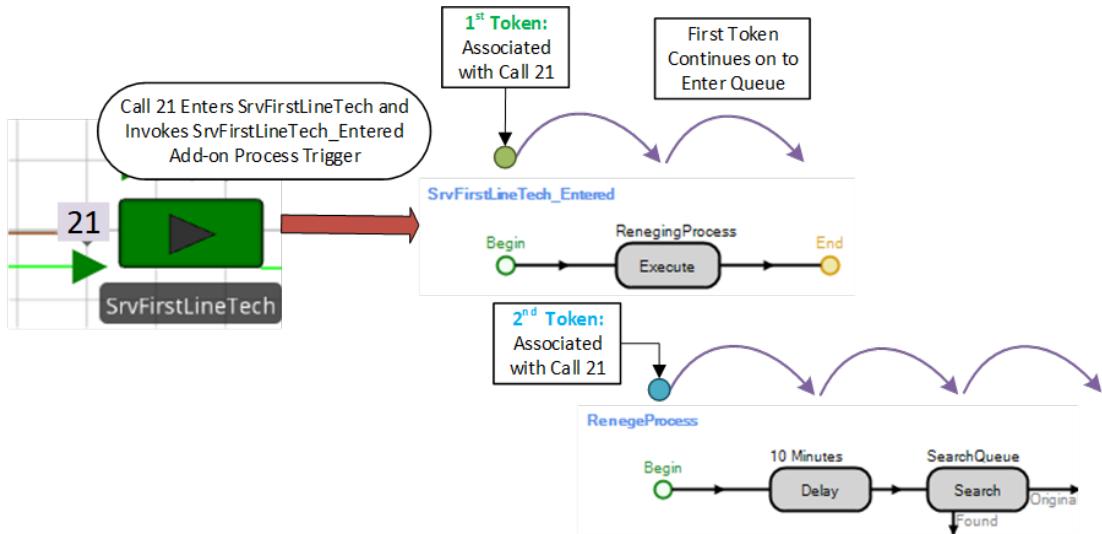
**Step 1:** Save the current model and then insert a new **SINK** named **SnkReneg** below the **SnkBalk**, but do not connect it to anything. This **SINK** will be used to collect statistics on those customers who renege.

**Step 2:** The modeling methodology to be used is when a call comes into the first-line technician's service queue. A ten-minute delay occurs, and then we check to see if the call is still in the queue. If it is in the queue and the position is greater than five, they renege; otherwise, you wait another four minutes to see if you have been served. Select the **SrvFirstLineTech SERVER** and insert the *Entered Add-on Process Trigger*.

*Question 14:* What would happen if we placed the delay and checked for reneging directly in this process?

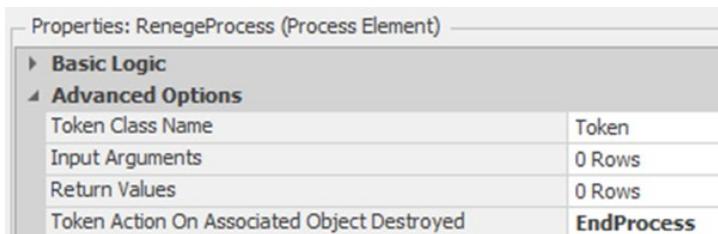
---

**Step 3:** The actual call **MODELENTITY** would wait ten minutes and then perform the check before entering the server to be placed into the queue or processed. Recall the discussion of how a token associated with the actual **MODELENTITY** invokes the process and executes the steps. In this case, we need one **TOKEN** to continue through the process (i.e., enter the queue and be processed) while a second **TOKEN** goes through the delay and for potential removal of the customer from the queue. Figure 16.10 illustrates how an *Execute* step can be used to spawn a new **TOKEN** associated (i.e., blue one) with the entity to execute another process. When the first **TOKEN** associated with the "Call 21" entity runs the *Execute* step, a new 2<sup>nd</sup> **TOKEN** is created that will execute the steps in the new **RenegProcess**. The first token can continue, allowing the call to enter the queue and possibly be served.



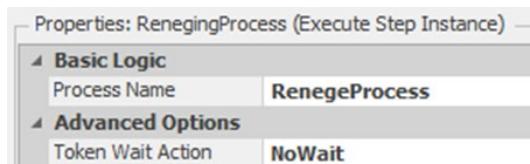
**Figure 16.10: Illustrate how Two TOKENS are Associated with the Same Call MODELENTITY**

**Step 4:** From the *Processes* tab, create a new process named **RenegeProcess**. Under the *Advanced Options*, set the *On Associated Object Destroyed* to **EndProcess**. Refer to Figure 16.11. If the MODELENTITY executing the process is destroyed, we want the **RenegeProcess** to end.<sup>255</sup> This will be important if it executes in parallel while the actual entity moves through the network.



**Figure 16.11: Ending the Process if the Entity has been Destroyed**

**Step 5:** Insert an *Execute* step into the **SrvFirstLineTech\_Entered** process, as seen in Figure 16.12 which executes the **RenegeProcess** process. Change the *Token Wait Action* property to “**NoWait**” which allows the TOKEN executing this step to continue on. By default it is set to wait until the **RenegeProcess** finishes, which would not allow the customer to continue to obtain service until the renege process ends (specify the *Execute* step as shown in Figure 16.12).

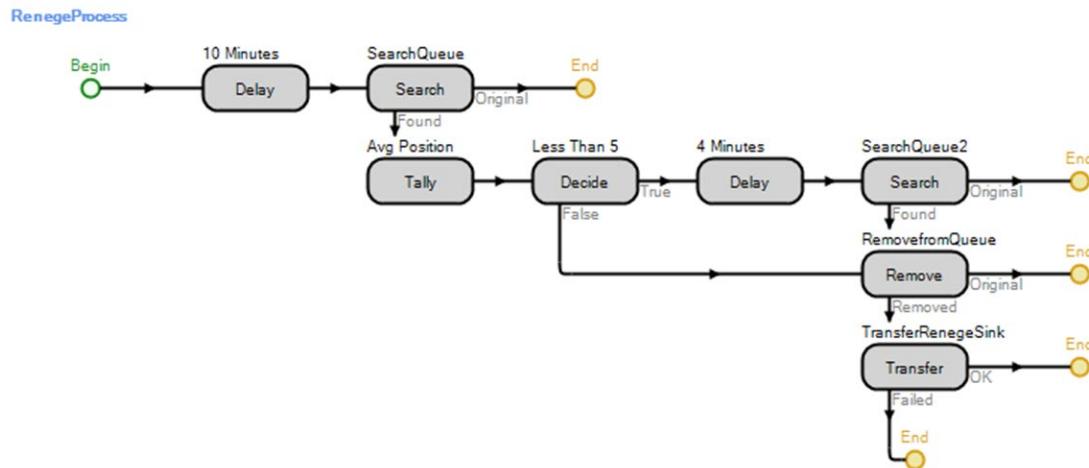


**Figure 16.12: Executing the Reneging Process but Allowing First Token to Continue**

**Step 6:** From the *Definitions* tab, add an **INTEGER DISCRETE STATE** variable named **GStaPosition** which will be used to store the position of the caller in the queue.

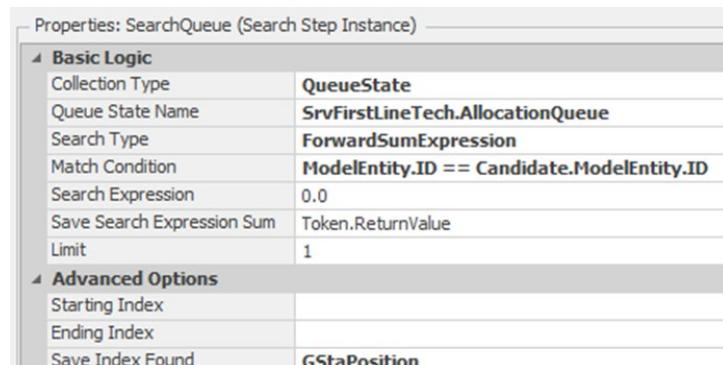
<sup>255</sup> Otherwise, the SERVER will be executing the process without the MODELENTITY and it will crash when performing the *Search* step.

**Step 7:** Return to the “Processes” tab and inset a *Delay* step into the **RenegeProcess**, as seen in Figure 16.13, which delays the TOKEN for ten minutes.<sup>256</sup>



**Figure 16.13: Process to Handle Reneging**

**Step 8:** After the ten-minute delay, we need to check to see if the associated call is still in the queue. If it is in the queue, determine its position. Insert a *Search* step after the *Delay* step, which will be used to search the queue and determine its position.<sup>257</sup> Change the *Collection Type* property to “QueueState” and the *Queue State Name* to **SrvFirstLineTech.AllocationQueue**. Specify the **GStaPosition** state variable as the *Save Index Found* property so it will be assigned the index of the item found. Since we want to search for the associated call in the queue, specify the *Match Condition* property as **ModelEntity.ID == Candidate.ModelEntity.ID**, which will search the queue and match the ID of the item in the queue with the associated entity ID, as seen in Figure 16.14.



**Figure 16.14: Specifying the Search Step Parameters to Search for the Call in the Service Queue**

**Step 9:** If the TOKEN associated with the **EntCalls** MODELENTITY finds itself in the queue, then the **GStaPosition** state variable needs to be checked to see if it is less than or equal to five. Therefore, insert a *Decide* step that performs a conditional test.

<sup>256</sup> Typically it is a good idea to create properties for values rather than hardcoding values which makes it easier to change later.

<sup>257</sup> See Chapter 13 for a discussion on using the *Search* step to look through collections.

Properties: Less Than 5 (Decide Step Instance)	
Basic Logic	
Decide Type	ConditionBased
Condition Or Probability	GStaPosition <=5

**Figure 16.15: Checking the Index of the Entity in the Queue**

**Step 10:** If the entity’s position in the queue is greater than five (i.e., the False branch), we will remove it from the queue and transfer it to the SINK **SnkReneg**. Insert a *Remove* step on the “False” branch that will remove the current call associated with the “Found” TOKEN from the SrvFirstLineTech.AllocationQueue.<sup>258</sup> You will need to type in the value for the *Queue State Name* property directly if it does not appear in the list.

**Step 11:** A TOKEN associated with the removed item will exit out the “Removed” path. Therefore, insert a *Transfer* step on the “Removed” path to move the entity from the current station to the **SnkReneg**.

Properties: TransferRenegSink (Transfer Step Instance)	
Basic Logic	
From	CurrentStation
To	Node
Node Name	Input@SnkReneg

**Figure 16.16: Transferring the Reneging Call to the Appropriate Queue**

**Step 12:** If the position in the queue is less than or equal to five, then delay the TOKEN for an additional four minutes and search again. Insert a *Delay* step on the “True” path that delays for four minutes.

**Step 13:** Copy the first *Search* step and place it after the four-minute delay to check and see if the entity is still in the queue.

**Step 14:** If the call is found in the queue (i.e., it is still waiting to be serviced), it will automatically renege. Grab the  (End) associated with the “Found” path and drag it on top of the *Remove* step to utilize the same *Remove* and *Transfer* steps as seen in Figure 16.13.

**Step 15:** For people who have been waiting for over ten minutes to talk with a technician, we would like to know their position after ten minutes on average. From the *Definitions*→*Elements* section, insert a new TALLY STATISTIC named **TallyStatPosition**. In the **RenegeProcess**, insert a *Tally* step right before the *Decide* step.

Properties: Avg Position (Tally Step Instance)	
Basic Logic	
Tally Statistic Name	TallyStatPosition
Value Type	Expression
Value Expression	GStaPosition

**Figure 16.17: Tracking Average Line Position**

**Step 16:** Save and rerun the Experiment.

**Question 15:** On average, how many calls renege during the 12-hour simulation run?

---



---

<sup>258</sup> Recall the *Remove* step can be used to remove entities from storage and allocation queues only.

*Question 16:* What is the average position for people who have waited at least ten minutes?

---

*Question 17:* What is the average time in the system for calls that don't balk or renege?

---

*Question 18:* What is the average time in the system for calls that renege?

---

## Part 16.4: Optimizing the Number of First-Line Technicians

We would like to determine if we have enough first-line technicians at the right times to minimize our customers' costs and wait times. Optimization with OptQuest™ was introduced earlier and will be used again. Sometimes, the WORKSCHEDULES may not be flexible enough, and you may need to resort to another technique. Nevertheless, at the end of the section, we return to specifying properties for the WORKSCHEDULE.

**Step 1:** Save the current model to a new name.

**Step 2:** From the *Definitions→Properties* section, insert 12 STANDARD EXPRESSION PROPERTIES with the names and values as seen Table 16.3.<sup>259</sup> These properties will be optimized to determine the best number to have during each time period, where TimePeriod1 would correspond to 8-9 AM.

**Table 16.3: Expression Property for the Time Periods**

Expression Property Name	Default Value	Category
TimePeriod1	5	Call Center
TimePeriod2	5	Call Center
TimePeriod3	8	Call Center
TimePeriod4	10	Call Center
TimePeriod5	10	Call Center
TimePeriod6	16	Call Center
TimePeriod7	16	Call Center
TimePeriod8	12	Call Center
TimePeriod9	8	Call Center
TimePeriod10	8	Call Center
TimePeriod11	8	Call Center
TimePeriod12	2	Call Center

**Step 3:** Next, insert a new DATA TABLE named **TableSchedule** under the *Data→Tables* section, which will become our own manual schedule table, as seen in Figure 16.18.

**Step 4:** Add a *Standard Expression* property column named **Capacity**.

---

<sup>259</sup> Create one property with the correct category, a name of **TimePeriod**, and then copy it changing the default value to the correct values.

**Step 5:** Insert 12 rows with the value of the expression for each row to be the corresponding property created above, as seen in Figure 16.18.<sup>260</sup>

Table Schedule	
	Capacity
1	TimePeriod1
2	TimePeriod2
3	TimePeriod3
4	TimePeriod4
5	TimePeriod5
6	TimePeriod6
7	TimePeriod7
8	TimePeriod8
9	TimePeriod9
10	TimePeriod10
11	TimePeriod11
12	TimePeriod12

Figure 16.18: Creating our own Work Schedule Table

**Step 6:** Since we are creating our own schedule, a TIMER will be needed to signal the time to change the capacity of the **SrvFirstLineTech** server at given intervals. From the *Definitions→Elements* section, insert a TIMER named **TimerSchedule** with a *Time Interval* of one hour and a *Time Offset* equal to zero.

**Step 7:** Under the *Definitions→States* section, insert a new DISCRETE INTEGER State variable named **GStaTimePeriod**, which will be used to set the current time period.

**Step 8:** Next, navigate to the *Processes* tab and create a new process named **ChangeCapacity** that responds to the **TimerSchedule.Event** as its *Trigger Event* property, as seen below.

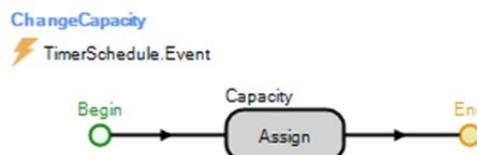


Figure 16.19: Changing the Capacity of the Timer Event

**Step 9:** Insert an *Assign* step that makes the following two assignments. The first assignment will assign **GStaTimePeriod** the value of zero through eleven by using the `Math.Remainder` function.<sup>261</sup> The next assignment will assign the current capacity of the **SrvFirstLineTech** SERVER a value from the table **TableSchedule**.<sup>262</sup> The simulation will start at `Run.TimeNow` is equal to zero, which is why we added one.

<sup>260</sup> The easiest way is to utilize Excel to create the 12 rows. First, add TimePeriod1 and TimePeriod2 into two different cells. Then select both of these cells and drag down letting excel create the other ten rows. Then copy and paste the Excel table into the **TableSchedule**.

<sup>261</sup> In this example, the remainder function takes care of the fact that `Run.TimeNow` will reach 12 and therefore the remainder will be zero and start over. If you are running 24 hours for more than one day then using `Math.Remainder(StaTimePeriod, 24)` is the way to get the correct time period for each day.

<sup>262</sup> Row indexes start at one and therefore we need to add one to the **StaTimePeriod** to get the correct period.

Basic Logic	
State Variable Name	GStaTimePeriod
New Value	Math.Remainder(Run.TimeNow,12)

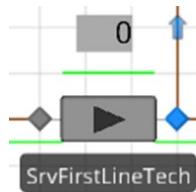
  

Basic Logic	
State Variable Name	SrvFirstLineTech.CurrentCapacity
New Value	TableSchedule[1 + GStaTimePeriod].Capacity

**Figure 16.20: Assignments to Perform our Manual WorkSchedule**

**Step 10:** Return the *Facility* tab and change the *Capacity Type* back to “Fixed” since we are performing the manual work schedule.

**Step 11:** In order to visually see if our new manual schedule is working, select the **SrvFirstLineTech**, and from the *Animation* tab, insert a new STATUS LABEL with an expression of **CurrentCapacity**.



**Figure 16.21: Inserting a Status to Watch the Current Capacity**

**Step 12:** Save and run the model, comparing the results to the previous model.

**Question 19:** Does the capacity of the server change during each hour’s break?

**Question 20:** Compare the results of the following questions to the previous model.

What is the average customer time in the system and the number of calls still in the system at the end of the simulation?

What is the average utilization of the first line and the experienced technicians?

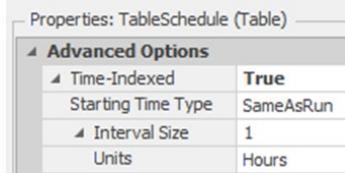
What is the average holding time in minutes for both of the two servers’ input buffers?

What is the average number of calls waiting in the two server’s input buffer?

How many calls balked on average during the 12-hour period, and do you feel the current system is handling enough of the calls?

**Step 13:** In the assignment, we manually converted the current simulation time into a row index (i.e., **GStaTimePeriod**), which, if we ran the simulation for 14 hours, would have started over at rows one and two for the 13<sup>th</sup> and 14<sup>th</sup> hour. DATA TABLES in SIMIO can be specified as being *Time-Indexed*, which allows one to access a column value based on the current simulation time. Select the table properties of the **TableSchedule** and specify under the *Advanced Options* that it is *Time-Indexed*. The starting time will tell SIMIO how to convert the current simulation time into which row, while the *Interval Size* determines the length of the time bucket to map to each row in the data table.<sup>263</sup>

<sup>263</sup> SIMIO is internally doing what we did to calculate the row index.



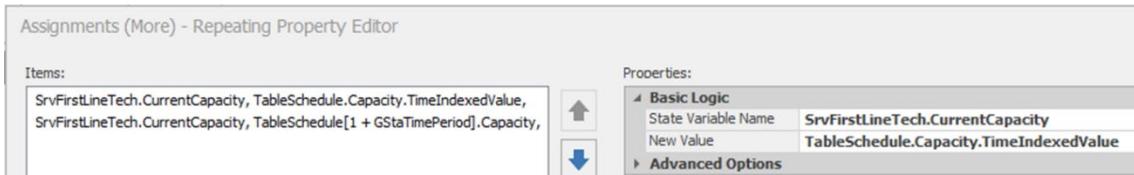
**Figure 16.22: Making a Data Table Time-Indexed**

Two table functions associated with time-indexed data tables can be used to retrieve column values or the current row index, as described in Table 16.4.

**Table 16.4: New Data Table Functions for Accessing Time-Indexed Columns**

Function	Description
TableName.ColumnName.TimeIndexedValue	The TimeIndexedValue will return the value of the column or property that has been specified for the current simulation time (i.e., TimedIndexedValue)
TableName.TimeIndexedRow	This function will return the current row index based on the current simulation time.

**Step 14:** From the *Processes* tab, modify the *Assign* step by removing the first assignment, which sets the **StaTimePeriod** and then sets the current capacity to *TableSchedule.Capacity.TimeIndexedValue*, as seen in Figure 16.23.



**Figure 16.23: Using the TimeIndexedValue Function to Set the Current Capacity**

**Step 15:** In the *Facility* window, insert two more status labels with expressions set to *TableSchedule.Capacity.TimeIndexedValue* and *TableSchedule.TimeIndexedRow* to see how these values change over time.

**Step 16:** Save and run the model, comparing the results to the previous model.

**Question 21:** Does the capacity of the server change at each hour break in a similar fashion?

**Question 22:** Compare the results of the following questions to the previous model.

What is the average customer time in the system and the number of calls still in the system at the end of the simulation?

---

What is the average utilization of the first line and the experienced technicians?

---

What is the average holding time in minutes for both of the two servers' input buffers?

---

What is the average number of calls waiting in the two server's input buffer?

How many calls, on average, balked during the 12 hours, and is this a good system now?

---

**Step 17:** Now that we have created a manual work schedule that can be optimized to demonstrate the use of time-indexed tables, we will utilize the built-in WORKSCHEDULE. From the *Data* tab, modify the WORKSCHEDULE to utilize all the time period properties for each of the values, as seen in Figure 16.24.<sup>264</sup>

The screenshot shows the 'Day Patterns' dialog box. Under 'StandardDay', there is a 'Work Periods' table with the following data:

Start Time	Duration	End Time	Value	Cost Multiplier
8:00 AM	1 hour	9:00 AM	TimePeriod1	1
9:00 AM	1 hour	10:00 AM	TimePeriod2	1
10:00 AM	1 hour	11:00 AM	TimePeriod3	1
11:00 AM	1 hour	12:00 PM	TimePeriod4	1
12:00 PM	1 hour	1:00 PM	TimePeriod5	1
1:00 PM	1 hour	2:00 PM	TimePeriod6	1
2:00 PM	1 hour	3:00 PM	TimePeriod7	1
3:00 PM	1 hour	4:00 PM	TimePeriod8	1
4:00 PM	1 hour	5:00 PM	TimePeriod9	1
5:00 PM	1 hour	6:00 PM	TimePeriod10	1
6:00 PM	1 hour	7:00 PM	TimePeriod11	1
7:00 PM	1 hour	8:00 PM	TimePeriod12	1

**Figure 16.24: Using Properties for the Values of a WORKSCHEDULE**

**Step 18:** Save and run the model, comparing the results to the previous model.

## Part 16.5: Using the Financial Costs as the Optimizing Objective

Since the TABLE to perform the task of developing a manual WORKSCHEDULE has been set up, we can use OptQuest™ to optimize the 12-time period properties. Our goal is now to minimize the costs associated with staffing the call center while balancing our customers' wait time and the amount of balking and renegeing that occurs to reflect customer dissatisfaction. SIMIO introduced ABC costing for many objects under the "Financials" property section. This cost information can be associated with dynamic (i.e., MODELENTITIES, TRANSPORTERS, and WORKERS) and fixed objects (i.e., SERVERS, SINKS, RESOURCES, etc.). Therefore, you do not have to capture this information using state variables manually. The following tables explain the financial cost properties of fixed objects (see Table 16.5), vehicle and worker objects (Table 16.6), and model entities (see Table 16.7).

---

<sup>264</sup> The easiest way to set the values is to copy them again from Excel which can be used to create the column.

**Table 16.5: Financial Properties Associated with Fixed Objects**

Property	Sub Property	Description
Parent Cost Center		The cost center element, which is the cost accrued to this server, is summed up. If the property is left blank, then the costs accrued are rolled up into the parent object containing the server (i.e., the MODEL).
Capital Cost		The one-time occurrence cost to build this fixed object occurs at the simulation's beginning.
Buffer Costs (Input, MemberInput, MemberOutput, ParentInput, ParentOutput and Output Buffer)	Cost Per Use	Costs associated with the buffers. This is a one-time use for every entity that enters the buffer regardless of how long they wait in the buffer (i.e., all entities will enter the input and output buffer regardless if they only stay in zero minutes).
	Holding Cost Rate	This is the cost rate applied per entity based on the amount of time the entity waits in the buffer.
Resource Costs (All fixed objects)	Idle Cost Rate	The cost charged per unit of time for any scheduled capacity of the object that is not utilized.
	Cost Per Use	This is a one-time use for every time the object is used, regardless of how long the object is used.
Resource Costs	Usage Cost Rate	The cost charged per unit of time for any scheduled capacity of the object that is utilized.

**Table 16.6: Financial Properties Associated with Vehicle and Worker Objects**

Property	Sub Property	Description
Parent Cost Center		See Table 16.5 for an explanation.
Capital Cost Per Worker/Vehicle		One-time occurrence cost to add this VEHICLE or WORKER.
Transport Costs	Cost Per Rider	The cost associated with loading/unloading and transporting an entity regardless of ride time.
	Transport Cost Rate	This is the cost rate applied per entity based on the amount of time the entity waits in the buffer.
Resource Costs	Idle Cost Rate	The cost charged per unit of time for any scheduled capacity of the server that is not utilized.
	Cost Per Use	This is a one-time use for every time the object is used, regardless of how long the object is used.
	Usage Cost Rate	The cost charged per unit of time for any scheduled capacity of the VEHICLE/WORKER that is utilized.

**Table 16.7: Financial Properties Associated with ModelEntities Objects**

Property	Description
Initial Cost	The initial cost for the creation of the MODELENTITY.
Initial Cost Rate	The cost charged per unit of time for any scheduled capacity of the server that is not utilized.

**Step 1:** Select the **SrvFirstLineTech** SERVER and expand the “*Financials*” property category. The first-line technicians receive \$25 per hour regardless of whether they are helping a customer or not. Therefore, set the *Idle* and *Usage Cost Rate* properties to “25,” as seen in Figure 16.25. Owing to the trunk line rentals, there

is a fixed cost (i.e., \$10) regardless of usage as well as a \$5 per hour per call usage rate that needs to be specified. We will leave the *Parent Cost Center* blank to utilize the MODEL's cost center.<sup>265</sup>

Financials	
Parent Cost Center	
Capital Cost	0.0
Buffer Costs	
Input Buffer	
Cost Per Use	10
Units	USD
Holding Cost Rate	5
Units	USD per Hour
Output Buffer	
Cost Per Use	0.0
Holding Cost Rate	0.0
Resource Costs	
Idle Cost Rate	25
Units	USD per Hour
Cost Per Use	0.0
Usage Cost Rate	25
Units	USD per Hour

**Figure 16.25: Specifying Cost Information on the SERVER**

**Question 23:** Save and run the model, observing the new cost values in the results section. What is the total cost of the system associated with the MODEL?

---

**Step 2:** Since our cost information is set up, we now need to optimize the number of technicians by minimizing the total costs associated with the system while balancing the number of unhappy customers. Create a “New Experiment” named **Optimization**.

**Step 3:** Add the following three responses to the experiment, which will determine the total cost of the system and keep track of the number of people who balk and renege.

Properties: TotalCost (Response)	Properties: Reneging (Response)	Properties: Balk (Response)																																																						
<table border="1"> <thead> <tr><th colspan="2">General</th></tr> </thead> <tbody> <tr><td>Name</td><td>TotalCost</td></tr> <tr><td>Display Name</td><td>TotalCost</td></tr> <tr><td>Description</td><td></td></tr> <tr><td>Expression</td><td>Cost</td></tr> <tr><td>Unit Type</td><td>Unspecified</td></tr> <tr><td>Objective</td><td>Minimize</td></tr> <tr><td>Lower Bound</td><td></td></tr> <tr><td>Upper Bound</td><td></td></tr> </tbody> </table>	General		Name	TotalCost	Display Name	TotalCost	Description		Expression	Cost	Unit Type	Unspecified	Objective	Minimize	Lower Bound		Upper Bound		<table border="1"> <thead> <tr><th colspan="2">General</th></tr> </thead> <tbody> <tr><td>Name</td><td>Reneging</td></tr> <tr><td>Display Name</td><td>Reneging</td></tr> <tr><td>Description</td><td></td></tr> <tr><td>Expression</td><td>SnkReneging.TimeInSystem.NumberObservations</td></tr> <tr><td>Unit Type</td><td>Unspecified</td></tr> <tr><td>Objective</td><td>Minimize</td></tr> <tr><td>Lower Bound</td><td></td></tr> <tr><td>Upper Bound</td><td>10</td></tr> </tbody> </table>	General		Name	Reneging	Display Name	Reneging	Description		Expression	SnkReneging.TimeInSystem.NumberObservations	Unit Type	Unspecified	Objective	Minimize	Lower Bound		Upper Bound	10	<table border="1"> <thead> <tr><th colspan="2">General</th></tr> </thead> <tbody> <tr><td>Name</td><td>Balk</td></tr> <tr><td>Display Name</td><td>Balk</td></tr> <tr><td>Description</td><td></td></tr> <tr><td>Expression</td><td>SnkBalk.TimeInSystem.NumberObservations</td></tr> <tr><td>Unit Type</td><td>Unspecified</td></tr> <tr><td>Objective</td><td>Minimize</td></tr> <tr><td>Lower Bound</td><td></td></tr> <tr><td>Upper Bound</td><td>15</td></tr> </tbody> </table>	General		Name	Balk	Display Name	Balk	Description		Expression	SnkBalk.TimeInSystem.NumberObservations	Unit Type	Unspecified	Objective	Minimize	Lower Bound		Upper Bound	15
General																																																								
Name	TotalCost																																																							
Display Name	TotalCost																																																							
Description																																																								
Expression	Cost																																																							
Unit Type	Unspecified																																																							
Objective	Minimize																																																							
Lower Bound																																																								
Upper Bound																																																								
General																																																								
Name	Reneging																																																							
Display Name	Reneging																																																							
Description																																																								
Expression	SnkReneging.TimeInSystem.NumberObservations																																																							
Unit Type	Unspecified																																																							
Objective	Minimize																																																							
Lower Bound																																																								
Upper Bound	10																																																							
General																																																								
Name	Balk																																																							
Display Name	Balk																																																							
Description																																																								
Expression	SnkBalk.TimeInSystem.NumberObservations																																																							
Unit Type	Unspecified																																																							
Objective	Minimize																																																							
Lower Bound																																																								
Upper Bound	15																																																							

**Figure 16.26: Add the Three Responses of Cost, Number Reneging, and Number Balking**

---

<sup>265</sup> A COST CENTER element can be specified to have the cost summed up. These cost centers can be specified in a hierarchical fashion (i.e., several objects roll up to one center for total costing (e.g., department costing), several centers can roll up to higher level centers (e.g., departments to a plant, plants to organizations, etc.). You can also use Assign steps to assign costs to centers dynamically.

**Step 4:** Like many problems, this is a multi-objective problem where the objectives often conflict (i.e., on the one hand, minimizing the number of technicians would minimize cost, while customers would like the maximum allowed at each period to minimize their wait time). As was done previously, we can convert the multi-objective by combining the objectives in a weighted fashion, which generally is not a bad idea.<sup>266</sup> Instead, we will utilize thresholds or bounds on the number of balking and reneging. Having 25 people balk or renege represents, on average, approximately 3% of the callers that are seen in the day, which is what management feels is adequate for customer satisfaction. We set the upper bounds of 15 and 10 for the number of people who are acceptable to balk and renege, respectively, as seen in Figure 16.26.

**Step 5:** From the *Add-Ins* section of the new experiment, select the OptQuest for SIMIO add-in. Next, for each of the 12 *Control* variables, specify the *Minimum Value* to be “1”, the *Maximum Value* to be “20,” and the *Increment* property value to be “1”. Properties of the model become *Control* variables automatically.

**Step 6:** Specify the **TotalCost** response as the *Primary Response*, as seen in Figure 16.27. Also, set the OptQuest™ parameter values based on the figure. We will need to run this optimization longer based on the number of variables and threshold constraints.

Primary Response	TotalCost
► Advanced Options	
▫ OptQuest for Simio - Parameters	
Min Replications	7
Max Replications	20
Max Scenarios	100
Confidence Level	95%
Relative Error	0.1
Objective Type	Single Objective

Figure 16.27: OptQuest™ Parameter Values

**Step 7:** Save and run the optimization experiment.

**Step 8:** Once the optimization is complete, filter the scenarios so that only the ones that are checked are present, representing the scenarios that meet the number of balker and reneging thresholds.<sup>267</sup> Next, sort the total cost column in ascending order. At this point, you should run the K-N selection algorithm on the top 20 or so scenarios to determine the best since only seven replications were used. Use a \$100 indifference zone.

*Question 24:* What did you get for the total cost of the best system?

---

*Question 25:* What did you get for the average number of balkers and reneges?

---

*Question 26:* What were the number of technicians needed for each time period?

---

<sup>266</sup> We could arbitrarily put a large cost on the input buffers of both the balk and renege SINKS to try to force the system to minimize the number of balking and reneging that occurs. However, determining the right value of the cost can be difficult.

<sup>267</sup> Select the filter option in the column heading row above the check boxes.

**Step 9:** The optimization has determined that more than 140 technician hours are needed based on costs. However, management only has access to 135 total hours. Add a constraint, which is a constraint on the decision control variables that utilizes the following expression, as seen in Figure 16.28.

TimePeriod1+TimePeriod2+TimePeriod3+TimePeriod4+TimePeriod5+TimePeriod6+  
TimePeriod7+TimePeriod8+TimePeriod9+TimePeriod10+TimePeriod11+TimePeriod12

Properties: TechnicianHours (Constraint)	
General	
Name	TechnicianHours
Expression	TimePeriod1 + TimePeriod2 + TimePeriod3 + TimePeriod4 + ...
Lower Bound	
Upper Bound	135

**Figure 16.28: Specifying Control Variable Constraint**

**Step 10:** Save and run the optimization experiment.

**Step 11:** Once the optimization is complete, filter the scenarios so that only the ones that are checked are present, representing the scenarios that meet the number of balker and reneging thresholds. Next, sort the total cost column in ascending order.

*Question 27:* What did you get for the total cost of the best system as well as the average number of balkers and reneges?

---

*Question 28:* What was the number of technicians needed for each time period, and what was the total number of technicians needed?

---

**Step 12:** Next, run the KN Select Best Scenario algorithm with an indifference zone of \$10 on the feasible solutions.

*Question 29:* What do you observe about the feasibility of the solutions now?

**Step 13:** Recall that the KN selection algorithm does not consider constraints on the non-primary objectives. Select the minimum one that meets the two objective thresholds.

*Question 30:* What did you get for the total cost of the best system and the average number of balkers and reneges?

---

*Question 31:* What was the number of technicians needed for each time period, and what was the total number of technicians needed?

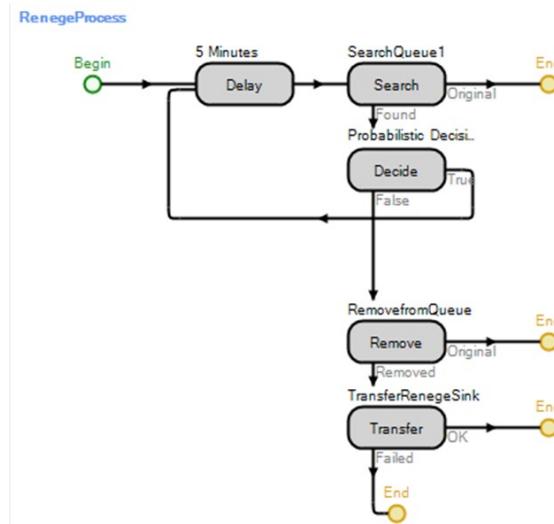
---

## Part 16.6: Commentary

- In this chapter, the *Execute* step was used to create a new **TOKEN** that worked independently of the original **TOKEN**. An additional modeling concept can cause the same phenomena to occur. The *Exited* add-on process triggers for objects can work in a similar fashion. The *Exited* add-on process triggers are executed when the trailing edge of the **MODELENTITY** has exited the node, link, station, or object,

depending on the trigger. Many people do not realize that the logic in these add-on processes runs in parallel while the entity continues to the link, station, etc. To illustrate this fact, you can clear the *Entered* Add-on process trigger for the **SrvFirstLineTech** and then select the Input node of the server. Set the *Exited* add-on-process trigger of the Input@**SrvFirstLineTech** BasicNode to be the **RenegeProcess** and rerun the model, comparing the results to the previous section.

- All of the SIMIO objects that Input and/or Output buffer stations have built-in Balking and Reneging options, which can handle very simple cases.
- More complicated reneging processes can also be utilized. For example, every five minutes, you are told how many people are still in line, and the person could make a probabilistic decision based on some behavior (i.e., number of people and how long they have already been waiting), as seen in Figure 16.29. In this example, if they search for the entity in the queue as before, it will decide to renege or not. If they decide not to renege (i.e., “True” path), it loops back and delays for another five minutes. The same remove and transfer is utilized if the caller decides to renege. The looping will continue until the caller no longer remains in the queue or reneges and leaves the system.



**Figure 16.29: Reneging Process that Checks Every Five Minutes**

# Chapter 17

## Sub-Modeling: Cellular Manufacturing

---

Object-oriented simulation languages allow users to extend the base language by creating new objects through composition and extension/inheritance. Most object-oriented languages offer the ability to create new objects out of existing ones, referred to as composition. SIMIO provides the ability to create objects via composition and extension. This chapter will explore the easier object creation (i.e., the composition of facility models inside other models (sub-models)).

A manufacturing operation has ten multiple work cells, and each work cell consists of two machines connected by a short conveyor, as seen in Figure 17.1. A dedicated operator is needed to load parts into the machine for processing on the first machine and then to unload them from the second machine. The conveyor system will automatically transfer the parts from the first machine to the next machine. Some work centers (i.e., machines within the work cell) can handle multiple parts in parallel. Also, priority is given to unloading parts from the second machine to ensure parts are processed through the cell quickly. One way to do this would be to insert ten Machine A servers, ten Machine B servers, ten conveyors, and ten fixed resources that are seized and released. Since the work cells are the same, we can use composition to create a work cell object and then use it nine times to model the whole manufacturing system.

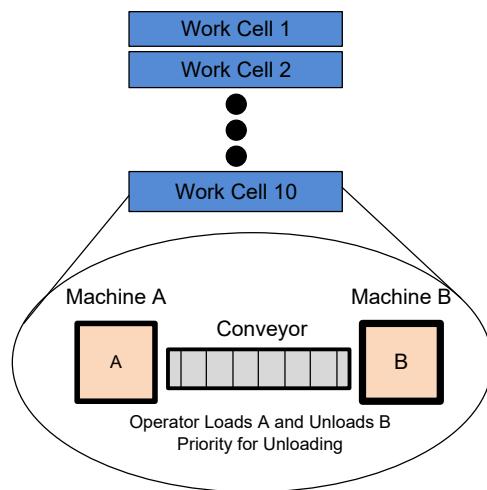
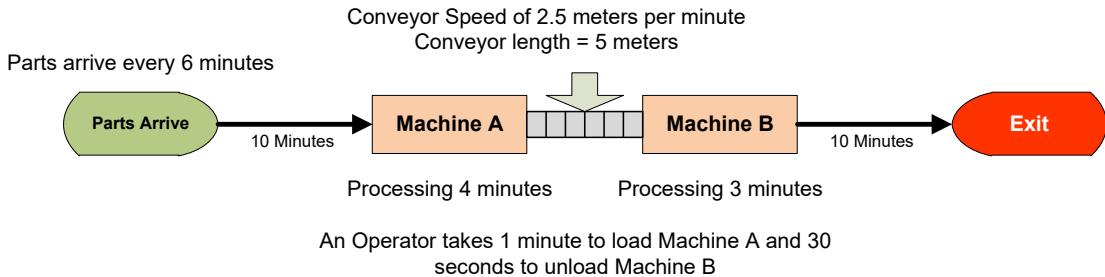


Figure 17.1: WorkCell Manufacturing Facility

### Part 17.1: Model of One Work Cell

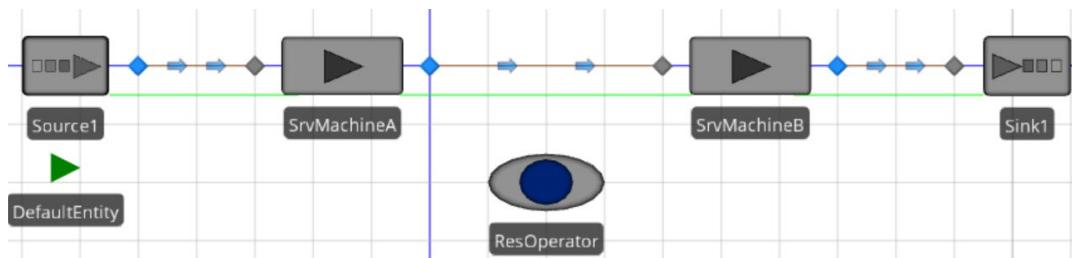
Before building the larger model, working initially with a single work cell is better. Eventually, this model will become a building block (i.e., composition object) and is referred to as a “sub-model” (see Figure 17.2).



**Figure 17.2: Individual Work Cell**

**Step 1:** Create a new project in SIMIO. Change the model name to **WorkCell** by changing the *Model Name* in the Model Properties.

**Step 2:** Construct the model according to Figure 17.3.



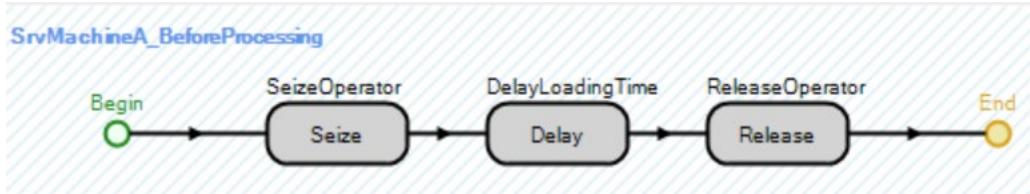
**Figure 17.3: Initial Two-Machine Model**

- Add two SERVERS named **SrvMachineA** and **SrvMachineB**. For now, the processing times should be four and three minutes.
- Connect these two machines with a five-meter conveyor with a speed of 2.5 meters per minute.
- Add a MODELENTITY, SINK, and SOURCE that generates arrivals every six minutes. Do not worry about naming since these are only temporary.
- Connect the SOURCE to **SrvMachineA**'s input and the sink to **SrvMachineB**'s output node via ten-minute time paths.

**Step 3:** Save and run the model to make sure entities flow correctly.

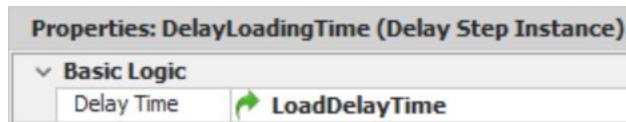
**Step 4:** The machines require an operator to load the part into the first machine, which generally takes one minute, and then to unload the part from the second machine, which takes approximately 30 seconds. Insert a new fixed RESOURCE named **ResOperator**. Add an additional symbol to indicate working and color the iris green.

**Step 5:** Add the “*Before\_Processing*” add-on trigger for Machine A that will seize the “Specific” resource **ResOperator**, delay for a certain amount of time, and then release the specific resource. This process represents parts that enter processing at Machine A but need to be first loaded into the machine by the operator before being processed.



**Figure 17.4: Process Logic to Seize and Release Operator to Load First Machine**

**Step 6:** The *Delay Time* property for the *Delay* step should be a referenced property. Right-click on the *Delay Time* and “Create a New Reference Property” named **LoadDelayTime**.



**Figure 17.5: Specifying a Referenced Property as the Delay Time**

**Step 7:** Repeat the procedure for unloading parts out of the second server **SrvMachineB**. The only difference is that the name of the referenced property should be **UnloadDelayTime**, and the add-on trigger should be the “*After Processing*” trigger (i.e., **SrvMachineB\_AfterProcessing**). Once the process has been created, copy and paste the process steps from the **SrvMachineA\_BeforeProcessing** and create the new reference property for the *Delay Time*.

**Step 8:** Switch to the “*Definitions*” tab, change the *Category* of the two new properties to “Process Logic,” and set the *Default Units* property to “Minutes.” The default value should be set to one minute for the **LoadDelayTime** property and 0.5 minutes for the **UnloadDelayTime** property.

**Step 9:** Save and run the model, making sure it behaves as intended.

*Question 1:* For 24-hour replication, what is the average time in the system (in minutes)?

## Part 17.2: Creating the Sub-Model

The primary work cell model works correctly, but we must create the sub-facility model to turn it into an object we can use in other simulation models. To create the sub-model, an *External* view of the sub-model must be defined before you can use it as an object in other models similar to the SERVER, SOURCE, etc.

**Step 1:** Delete the SOURCE, SINK, and MODEL ENTITY from this facility model since they were only needed to test the logic of the two machines connected via a conveyor.

**Step 2:** Since the **WorkCell** model will be used as a sub-model, none of the nodes in the model (i.e., **Input@SrvMachineA** or **Output@SrvMachineB**) are visible to the external model (i.e., models using the **WORKCELL** object). Therefore, two external nodes need to be added to the external view of the object (i.e., *Definitions*→*External* section) to facilitate the flow of entities into and out of the sub-model.<sup>268</sup>

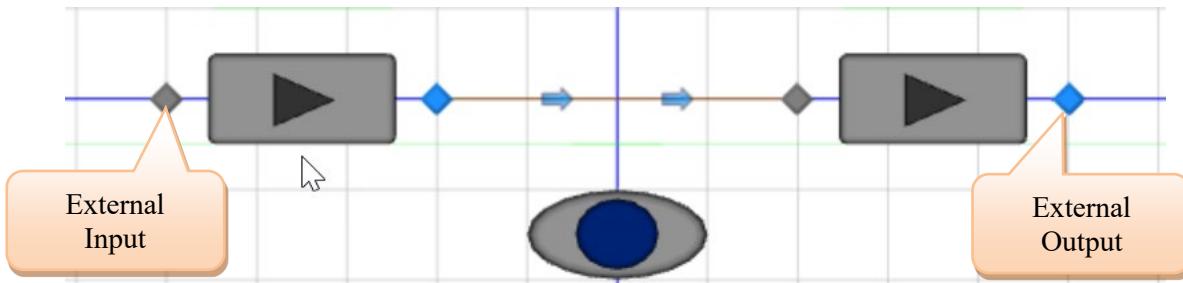
<sup>268</sup> All the basic objects (e.g., SOURCE, SERVER, WORKSTATION, etc.) have nodes in their external view that allow entities to flow into and/or out of the object.

- Since entities entering the **WORKCELL** object need to enter the input node of machine A (i.e., **Input@SrvMachineA**), right-click the **BASICNODE** to bring up the sub menu seen in Figure 17.6 and select “Bind to New External Input Node” and then name the external node **Input** to be consistent.<sup>269</sup>
- Next, right-click the output node of **SrvMachineB**, but this time select “Bind to New External Output Node” and name it **Output**.
- By default, all objects (i.e., animated queues, objects, etc.) in the facility are visible to the external world as indicated by the “Externally Visible” button, which is toggled on, as seen in Figure 17.6. Toggle the *externally visible* button off for both the **Input@SrvMachineA** and **Output@SrvMachineB** nodes. This will make it easier to see our new **Input** and **Output** nodes.



**Figure 17.6: Automatically Adding External Nodes as Making Objects Visible to the External View**

**Step 3:** Navigate to the *External* panel underneath the *Definitions* tab and survey the external view, which includes the two external nodes named **Input** (a **BASICNODE**) and **Output** (a **TRANSFERNODE**)<sup>270</sup>, as well as a symbol representing the model, as seen in Figure 17.7. Only the two nodes are accessible.



**Figure 17.7: External View of the WorkCell Object**

**Step 4:** Select the **Input** node, which can be seen as an instance of the **BASICNODE** NODE Class. Generally, entities are transferred from this input node to a station. However, the sub-facility model does not expose a station. Therefore, these entities will be transferred to the input of the **SrvMachineA**. The *Input Location Type* property is set to “FacilityNode,” and the *Node Name* is the **Input@SrvMachineA** as in Figure 17.8. As entities flow into the **Input** of the **WorkCell**, they will be automatically routed to **SrvMachineA**.

<sup>269</sup> These can be added manually from the *External* panel underneath the “*Definitions*” table however all of the property settings also have to be done manually.

<sup>270</sup> The **Input** and **Output** nodes are on top of the picture of the nodes associated with the **SrvMachineA** and **SrvMachineB**. The model in the *External* view is only a picture of what the **WorkCell** will appear in other models.

Properties: Input (ExternalNode Instance)	
Initial Property Values	0 Rows
Node Class Name	BasicNode
Input Location Type	Node
Node Name	Input@SrvMachineA
Name	Input

Figure 17.8: The Input Node

**Step 5:** After selecting the **Output** node, you can see this is an instance of the TRANSERNODE class to allow entities to be routed to their next destination once they leave the **WorkCell** (see Figure 17.9).

Properties: Output (ExternalNode Instance)	
Initial Property Values	0 Rows
Node Class Name	TransferNode
Input Location Type	None
Name	Output

Figure 17.9: The Output Node

**Step 6:** Entities that enter the external input node of the **WORKCELL** object will automatically be transferred to the input of the **SrvMachineA** via the **Input** node. When entities reach the output node of the **SrvMachineB**, they need to be transferred to the external output node (**Output**) defined in the external view. Switch back to the “Facility” tab and select the Output node of **SrvMachineB** (i.e., **Output@SrvMachineB**). Under the *Advanced Options* category, you can see that SIMIO automatically changed the *Bound External Output* property to specify the **Output** as the external node name property, as shown in Figure 17.10. Now, entities will automatically be transferred to the external **Output** when they reach the output of the **SrvMachineB**.

Display Name	
Outbound Link Random Number Stream	0
Sequence Expected Operation Time	0.0
Bound External Output Node	Output

Figure 17.10: Connecting the External Node

**Step 7:** Save the **WorkCell** model.

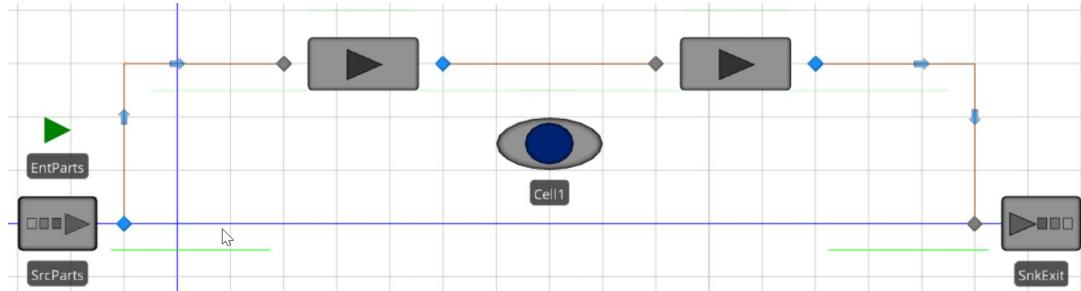
### Part 17.3: Creating a Model using the WorkCell Sub-model

The **WORKCELL** Sub-facility model has been built and can now be saved to a library and used as any other object that is part of SIMIO. We are going to develop our manufacturing systems using the **WORKCELL**.

**Step 1:** From the *Project Home* tab *Create* section, insert/create a new “Fixed Class” model.

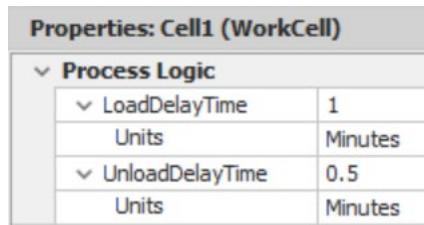
**Step 2:** In the new model, insert a SOURCE named **SrcParts**, a SINK named **SnkExit**, and a new work cell named **Cell1**, which is part of the *Project Library* panel. Connect the SOURCE and the SINK to the **WORKCELL** via ten-minute time paths, as seen in Figure 17.11.

**Step 3:** Make the interarrival time of parts is Exponential with a mean of 5 minutes.



**Figure 17.11: Simulation Model Using the New WorkCell Sub-Simulation Model**

**Step 4:** For **Cell1**, ensure the *Load Delay Time* property is one minute while the *UnloadDelay* time property is 0.5 minutes.



**Figure 17.12: Setting the Properties of the WorkCell**

**Step 5:** Save and run the model for 24 hours, looking at the results. Notice that the statistics of all the pieces of the work cell are visible under the **WORKCELL** object type.

**Question 2:** For a single 24-hour replication, what is the average time in the system (in minutes)?

---

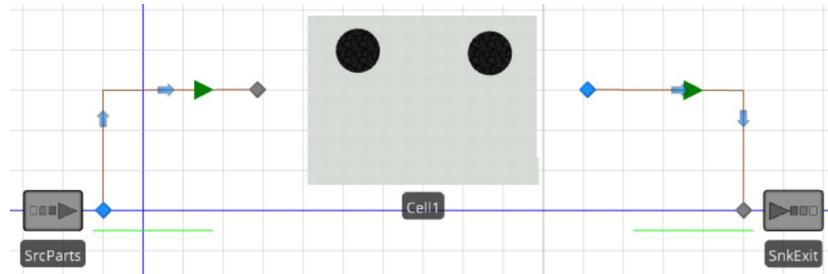
**Question 3:** From an animation standpoint, what did you observe when the parts entered the workcell?

---

**Step 6:** Select the **WORKCELL** **Cell1** and choose a different symbol for animation purposes.<sup>271</sup>

---

<sup>271</sup> In this example, a factory symbol was chosen and it was resized and rotated.



**Figure 17.13: Changing the Symbol**

*Question 4:* What did you observe when the parts entered the workcell?

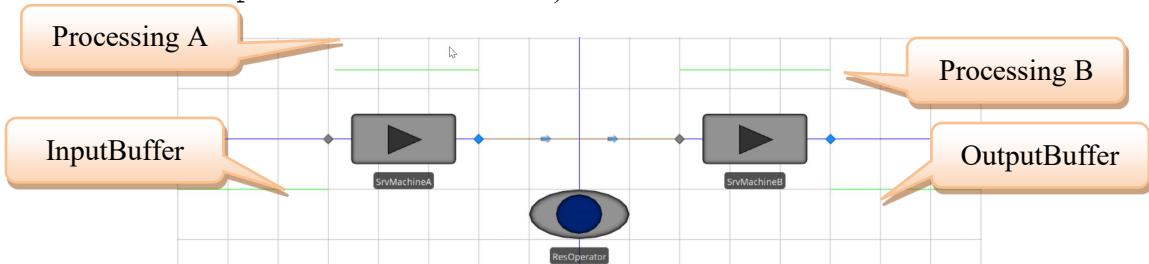
---

#### Part 17.4: Adding to the WorkCell Object

The internal view of the composition object (i.e., sub-model) is the object's symbol, which removes the internal workings when replaced with a new symbol. In this simple example, showing the internal workings may make sense, but hiding the details of the sub-model in more complicated sub-models may be more beneficial (i.e., just having a plain symbol to represent the object). However, when using a different symbol, it would be helpful to see some indication of the locations of the entities (i.e., animated queues).

**Step 1:** Select the **WORKCELL** object in the [Navigation] panel and select the “*Facility*” tab. Select all of the animated queues and right-click to toggle off the *Externally Visible* option.

**Step 2:** Go to the eternal view under the “*Definitions→External*” section; add four animation queues from the “*Animation*” tab on the ribbon for the new work cell object. One queue should animate the **SrvMachineA**.**InputBuffer**.**Contents** to represent entities waiting to be processed while both processing queues of each server (**SrvMachineA**.**Processing**.**Contents** and **SrvMachineB**.**Processing**.**Contents**) should be animated. Finally, the output buffer queue of the **SrvMachineB** is animated to animate entities waiting to leave the work cell (**SrvMachineB**.**OutputBuffer**.**Contents**).<sup>272</sup>



**Figure 17.14: External View Showing the Four Animated Queues**

**Step 3:** These four queues will remain visible when the **WorkCell**'s animation symbol changes. Navigate back to the original model. The new animated queues will not be visible on objects already declared in the model. You can manually add these to the current **Cell1** object or delete it and then add it back to the model making sure to specify the 10-minute paths and changing the symbol, etc.

---

<sup>272</sup> Draw the queues from right to left because the first point drawn will be the start/head of the queue.

*Question 5:* What did you observe when the parts entered the work cell?

**Step 4:** Run the model by observing the animation of the WorkCell when a different symbol is used.

Currently, the entity will be hidden from the animation view when it is on the conveyor and waiting for processing at machine B or any other place with an animated queue associated with it. Since the animation of the internal submodel is not shown, we can reveal some of the behavior by placing labels that display the number of entities at the various locations. However, it would be helpful to have a representation of the number of parts currently being processed by the work cell either by machine A or B, as well as the number traveling on the conveyor and those waiting in the input buffer of machine B.

**Step 5:** SIMIO has the ability to specify user-defined storages (queues). Select the **WorkCell** model in the [Navigation] panel and navigate to the *Definitions* tab. Under the *Elements* section, insert a new **STORAGE** element named **StorageWorkCellInProcess**. We will accept the default ranking rule of “FirstInFirstOut”.

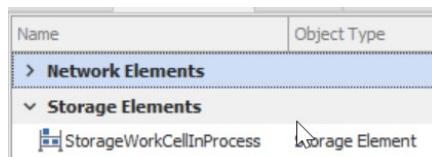


Figure 17.15: Adding a Storage

**Step 6:** Recall there are two process steps (i.e., *Insert* and *Remove*) that can be used to manipulate user-defined queues.<sup>273</sup> Once the part enters processing at **SRVMACHINEA**, it should be inserted into the **StorageWorkCellInProcess** queue to indicate they have gone into processing. Add an *Insert* step after the operator is released in the processing with the *Queue State Name* property associated with the new user-defined queue **WorkCellInprocess.Queue**. The *Object Type* should remain the “AssociatedObject” because the token is associated with the part.<sup>274</sup>

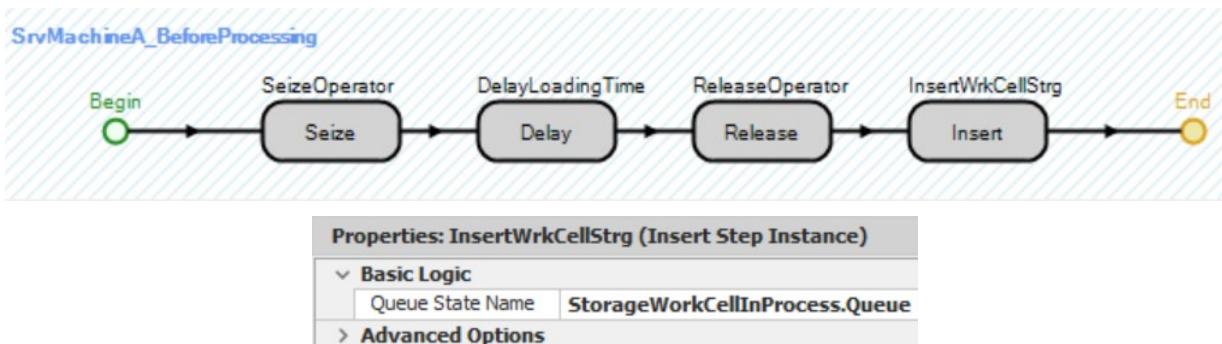
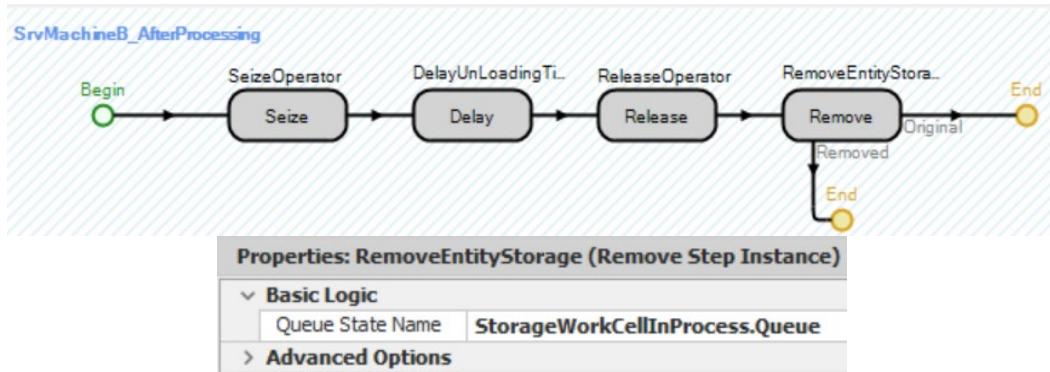


Figure 17.16: Using an **Insert** Step to Add Parts to a User-Defined Queue

**Step 7:** Once the part has finished processing and removed from the machine, delete the part from the user-defined queue using a *Remove* step, as seen in Figure 17.17.

<sup>273</sup> These steps do not work on queues defined by SIMIO (which is quite a limitation).

<sup>274</sup> The *Rank Placement* property can be used to specify the placement of the entity in the queue ignoring the overall ranking of the queue.



**Figure 17.17: Using a Remove Step to Delete Parts to a User-Defined Queue**

**Step 8:** Go to the *Definitions*→*External* view section for the **WorkCell** Object and add an Animation Queue above the object that animates the **StorageWorkCellInProcess.Queue** and delete the **Processing.Contents** queues for machines A and B that were inserted earlier.

**Step 9:** Return to the main model and manually insert an animated queue for **Cell1**, namely **Cell1.StorageWorkCellInProcess.Queue**. Recall none of the external view changes will appear for objects already inserted.

**Step 10:** To see that the model is working correctly, insert five additional symbols for the Part's coloring, each one with a different color. Set the *Animation*→*Random Symbol* property of the **entParts** to "True." Change the mean interarrival time to be Exponential(2) minutes for the **SrcParts**.

**Step 11:** Save and run the model, observing that the parts are now animated even if they are currently on the conveyor. The object now acts like a server with an input buffer, processing queue (i.e., **StorageWorkCellInProcess**), and an output buffer.

**Question 6:** For 24-hour replication, what is the average time in the system (in minutes)?

---

## Part 17.5: Exposing Resource and Capacity Properties

Currently, the operator is hidden inside the **WORKCELL** sub-facility model and, thus, cannot respond to work schedules. Also, the capacity and processing times of the two machines are hidden from the user of the objects.

**Step 1:** Return to the **WORKCELL** model and select the RESOURCE **ResOperator**. All fixed models have built-in properties of capacity type, capacity, ranking rule, and dynamic selection rules. Therefore, the **ResOperator** should be set equal to the **WorkCell** object's properties. Select the **ResOperator** and specify all of the process logic properties to their associated **WorkCell** properties, as seen in Figure 17.18. When the user specifies these properties of the **WorkCell**, the **ResOperator** properties will be set to the same.

Properties: ResOperator (Resource)	
Resource Logic	
Capacity Type	CapacityType
Initial Work Schedule	InitialWorkSchedule
Work Day Exceptions	WorkDayExceptions
Work Period Exceptions	WorkPeriodExceptions
Initial Capacity	InitialCapacity
Ranking Rule	RankingRule
Ranking Expression	RankingExpression
Dynamic Selection Rule	DynamicSelectionRule

**Figure 17.18: Exposing the Resource Capacity and Selection to the User**

**Step 2:** Each of the fixed model inherited properties by default is hidden. Therefore, you will need to go to the *Definitions→Properties* section, expand down the *Properties (Inherited)*, and set the *Visible* property to “True” for all the properties in Figure 17.18 as well as change the *Category* to “Process Logic.”

**Step 3:** Create a new reference property for the *Processing Time* property for each of the two servers named **MachineAProcessingTime** and **MachineBProcessingTime**, respectively, and the *Initial Capacity* property named **MachineAInitialCapacity** and **MachineBInitialCapacity**.

**Step 4:** In the *Definitions→Properties* section, place all of these new properties into the “Process Logic” category and set the default units of the processing times to minutes.

Properties: MachineBProcessingTime (Expression Property)	
Value	
Default Value	Random.Triangular(.1,.2,.3)
Switch Property Name	
Switch Condition	Equal
Switch Value	
Candidate References	False
Unit Type	Time
Default Units	Minutes
Appearance	
Display Name	MachineBProcessingTime
Category Name	Process Logic
Expanded	False
Parent Property Name	
General	
Name	MachineBProcessingTime

**Figure 17.19: Machine B Processing Time Property Values**

**Step 5:** Return the main model, making sure to set the processing times of machines A and B to be equal to four and three minutes, respectively, if they are not already.

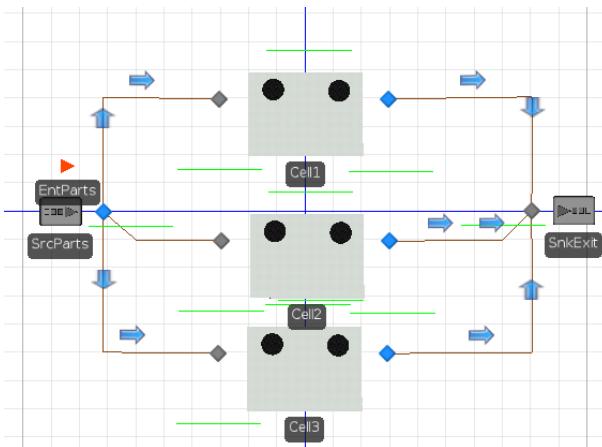
**Step 6:** Save and run the model, observing that nothing has changed.

**Step 7:** If you were only going to have one cell, then it does not make sense to take the time to create a sub-model object. However, the benefit of a sub-model object is that it can be reused repeatedly without repeating the same process logic and properties. Another advantage is if the logic changes or additional capabilities are needed, then changes need to be made in only one place.

**Step 8:** Insert two new **WORKCELL** objects named **Cell2** and **Cell3** by copying the **Cell1** object to include the same symbol with the same processing times of four and three minutes but make the capacities equal to two for the capacities for the two machines. Notice that the new animated queue has been automatically added.

**Step 9:** Connect the SOURCE and SINK via five-minute time paths to these new work cells and increase the arrival rate to two per minute.

**Step 10:** Save and run the model, observing the results.



**Figure 17.20: Demonstrating the Reuse of the WorkCell Object**

**Step 11:** Next, create a DATA SCHEDULE named **SchedCell** under the Workschedules view of the *Data* tab. Change the Standard Day such that there is a repeating schedule with three hours being “Off Shift” and three hours being “On Shift.” “On-shift” should have a capacity of “4” as seen in Figure 17.21.

**Step 12:** Select all three work cells and specify the *Capacity Type* property to follow a WORKSCHEDULE with the *Work Schedule* property the **SchedCell** as seen below.

Work Schedules					
Name	Start Date	Description	Days	Monday	Tuesday
↳ SchedCell	1/3/2011	Standard Work Week Schedule	7	StandardDay	StandardDay

Day Patterns					
Name	Description				
↳ StandardDay	Standard 8-5 Work Day				
Work Periods					
Start Time	Duration	End Time	Value	Cost Multiplier	Description
12:00 AM	3 hours	3:00 AM	4	1	
6:00 AM	3 hours	9:00 AM	4	1	
12:00 PM	3 hours	3:00 PM	4	1	
6:00 PM	3 hours	9:00 PM	4	1	

Properties: Cell1,Cell2,Cell3 (Multiple Objects)	
Process Logic	
Capacity Type	WorkSchedule
Initial Work Schedule	SchedCell
Work Day Exceptions	0 Rows
Work Period Exceptions	0 Rows
Ranking Rule	First In First Out

**Figure 17.21: Specifying the WorkCells follow a WorkSchedule**

**Question 7:** What do you notice about the parts in the cell that are in process when the shift changes?

---

**Question 8:** For 24-hour replication, what is the average time in the system (in minutes)?

---

## Part 17.6: Passing Information between the Model and its Sub-Models

Sometimes, the sub-model must be constructed using information from the model before the model has been constructed. For example, suppose the **WorkCell** needs an operator for **MachineA**, but the main model supplies the operator and is unknown to the sub-model. The sub-model must be created knowing that an operator will be available, but it doesn’t know its exact specifications. Here, a property “reference” is useful for the sub-model, expecting the property instance to be supplied in the model.

**Step 1:** Using the model from the previous section, select the **WorkCell** from the [Navigation] panel. From the *Definitions*→*Properties*, add a new Object List Property from the Object Reference dropdown named **ResourceList** and specify the “Process Logic” category. Set the Required Value property to **False** since the main model will supply the actual list.

**Step 2:** Click on the **SrvMachineA** server and use the *Secondary Resources* section using the *Resources for Processing* specification to select a resource randomly from the **ResourceList**, as shown in Figure 17.22.

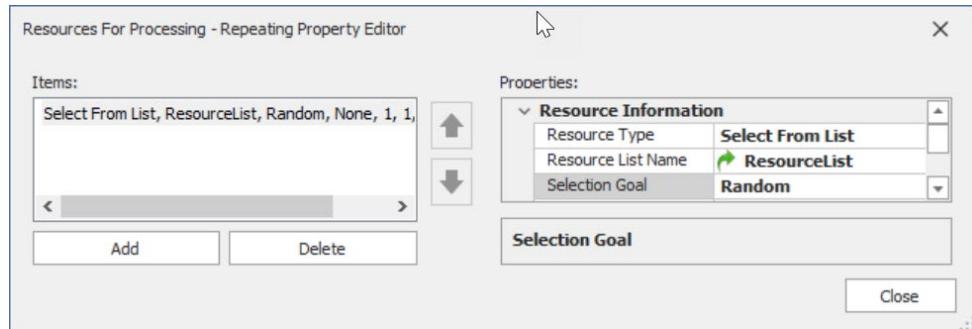


Figure 17.22: Specifying the Resource

**Step 3:** Select the **MODEL** and add two resources named **ResOperator1** and **ResOperator2**. Add a new list of objects within the *Definitions* tab and the *Lists* option. Call it **GResourceList** and have it contain the two operators.

**Step 4:** Specify the **GResourceList** for the individual cell’s *ResourceList* property. Doing this will link the model list to each of the sub-model lists.

**Step 5:** Run the model to see if it appears to be executing appropriately.

**Step 6:** Currently, we do not know where the operators are working since they are working at the sub-model level. At the model level, add an **INTEGER STATE VARIABLE** to the **MODELENTITY** called **EStaCellNumber**, which will be used to identify the cell.

**Step 7:** In the *State Assignments* section of the **TIMEPATHS** between the **SrcParts** and the cells, use the **Before Exiting** specification to assign the cell number to the **EStaCellNumber**. Figure 17.23 shows the assignment on the path to Cell 3. Repeat for the other two cells.<sup>275</sup>

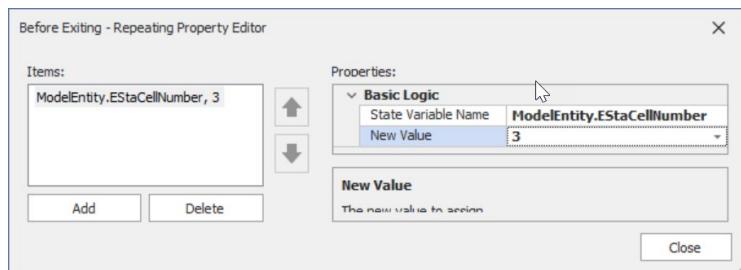
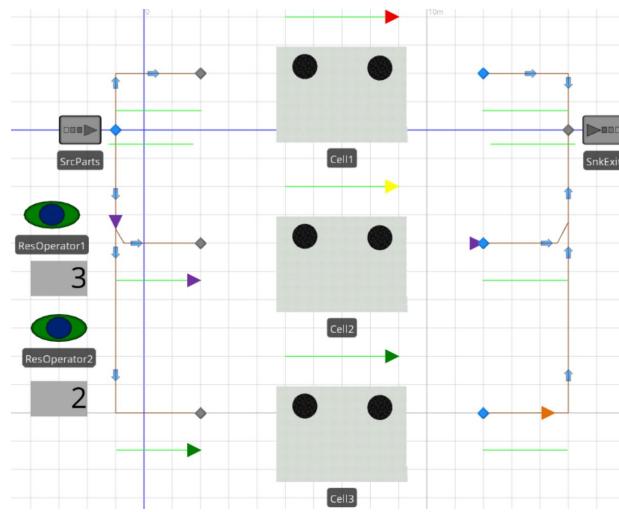


Figure 17.23: Assigning the Cell Number

<sup>275</sup> This is another great place to use a Tokenized process to set the cell number instead of using *Before Exiting* assignments especially if the number of work cells increased way beyond three.

**Step 8:** Add *Status Labels* under each resource to display which cell the resource is working (refer to Figure 17.24). Show the status of each of the resources. The following expression is for **ResOperator1**, which first checks to make sure the RESOURCE has been seized and, if it has, will display the entity's cell number.

```
Math.If(ResOperator1.ResourceOwners.NumberItems > 0,
      ResOperator1.ResourceOwners.FirstItem.ModelEntity.EStaCellNumber, 0)
```



**Figure 17.24: Communicating With the Sub-Model**

**Question 9:** Is the model executing as expected?

---

### Part 17.7: Commentary

- The idea behind the sub-model is that you can create your own object class by “composing” a model and then using it as an object. Once that object class is created/defined, you can use the object from that class multiple times for the same project or for different projects.

# Chapter 18

## The Anatomy of Objects: Server

Many simulation languages do not have the ability to modify the behavior of the objects and thus require another object to perform supporting activities (reneging is an example). Unlike most simulation languages, if you do not like the way the designers of SIMIO implemented the SERVER object (or any other SIMIO object), you can modify it or create your own object.

### Part 18.1: A Simple Resource Model: Warehouse Pickup

A warehouse store has a pickup area where customers come for items that are not stored on shelves. The pickup area is serviced by an associate who waits on those customers. Because the pickup area does not require the full attention of the associate, the associate is also asked to do some specialty stocking. Quite appropriately, the store wants the customers serviced before doing any stocking.

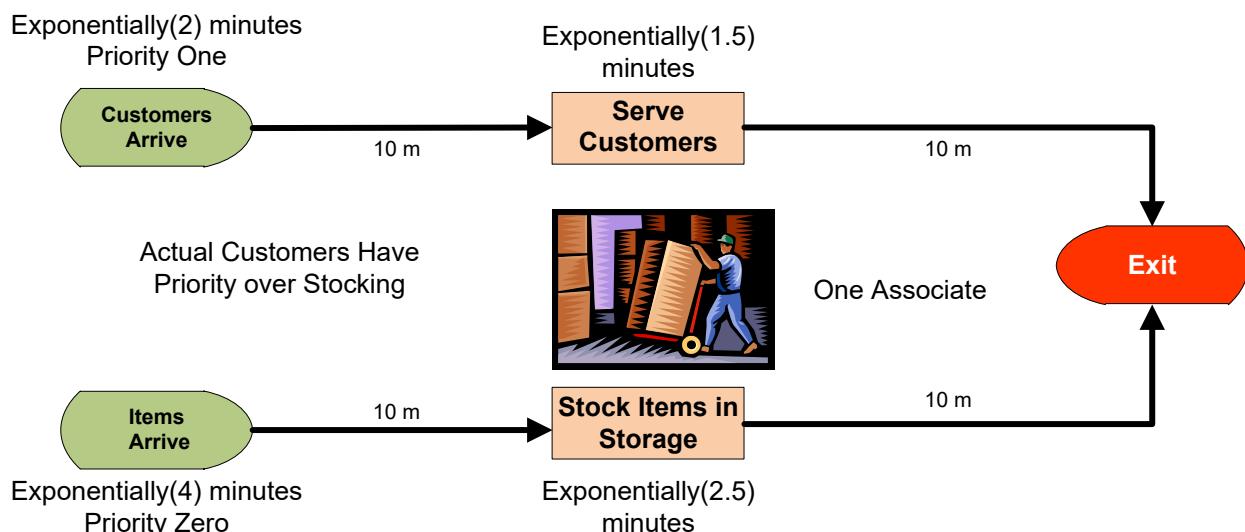


Figure 18.1: Warehouse Operations

**Step 1:** Create a new model.

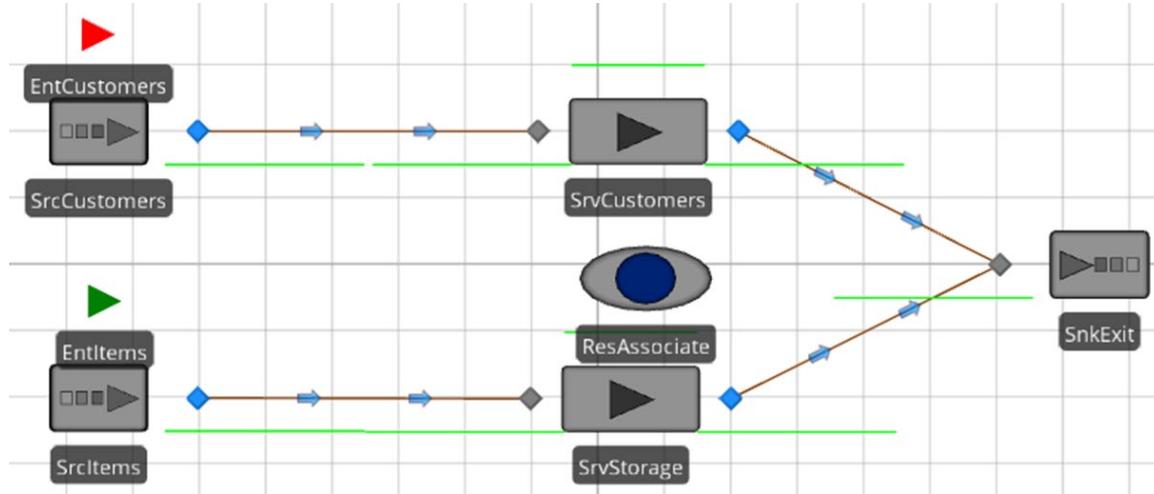
**Step 2:** Insert two SOURCES named **SrcCustomers** and **SrcItems** and two MODELENTITIES named **EntCustomers** and **EntItems**.

- The **SrcCustomers** should create **EntCustomers**, while the **SrcItems** create **EntItems**.
- Customers arrive exponentially every two minutes, while items arrive exponentially every four minutes.
- Color the **EntCustomers** red.
- Set the *Initial Priority* to zero for **EntItems** since, by default, MODELENTITIES have a priority of one.

**Step 3:** Insert two SERVERS named **SrvCustomer** and **SrvStorage** and one SINK named **SnkExit**.

- Each of the SERVERS should have a capacity of one.
- The *Processing Time* property for **SrvCustomer** should be set to Exponential (1.5) minutes and Exponential (2.5) for **SrvStorage**.

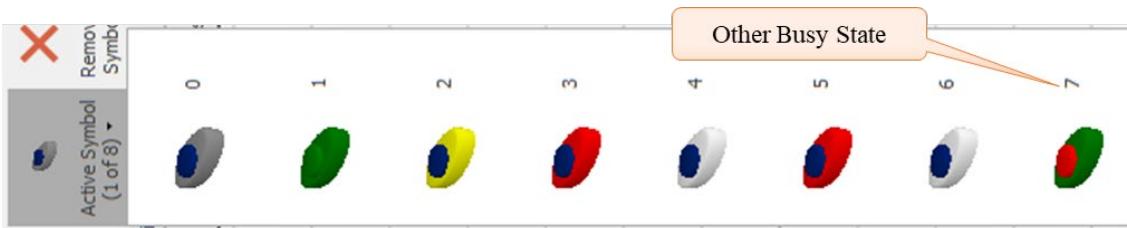
- Connect the two SOURCES to their appropriate SERVER via 10-meter logical paths and the two SERVERS to the SINK via ten-meter paths as seen in Figure 18.2.



**Figure 18.2: Model of a Warehouse Pickup Stations**

**Step 4:** Insert a fixed RESOURCE named **ResAssociate** between the two SERVERS, as seen in Figure 18.2, to constrain the processing of entities.

- Change the **ResAssociate's Dynamic Selection Rule** property to be “Largest Value First” with the *Value Expression* left as the default property (i.e., Candidate.Entity.Priority). This should force the associate to service the customers first if there is a choice.
- Recall the RESOURCE, which has seven defined states defined by the ResourceState variable. In order to visually indicate the type of part that is currently being served, add an additional symbol coloring the round circle red and the outside green for the 8<sup>th</sup> symbol to represent busy for red parts. Then select the 1<sup>st</sup> symbol and color the iris green, as seen in Figure 18.3.



**Figure 18.3: Adding a Second Symbol for Associate**

**Step 5:** From the *Definitions→States* section, add a new DISCRETE INTEGER STATE variable named **GStaAssociatePicture** to be used to specify the symbol associated with the customer type since the RESOURCE picture cannot be specified directly.

**Step 6:** Change the *Current Symbol Index* of the property of the **ResAssociate** to use the new state variable, as seen in Figure 18.4. If the state is busy and the entity is a customer, then display the 7<sup>th</sup> symbol; otherwise, show the default state symbol.

Animation	
Current Symbol Index	Math.If(Resource.ResourceState==1 && GStaAssociatePicture == 1,7, Resource.ResourceState)
Random Symbol	False

Figure 18.4: Animation Symbol Index

**Step 7:** From the *Processes* tab, create two new processes named **SeizeAssociate** and **ReleaseAssociate**. The **ReleaseAssociate** process should release the associate while the **SeizeAssociate** should seize the associate as well assign the **GStaAssociatePicture** the value of the **ModelEntity.Priority** as seen in Figure 18.5.

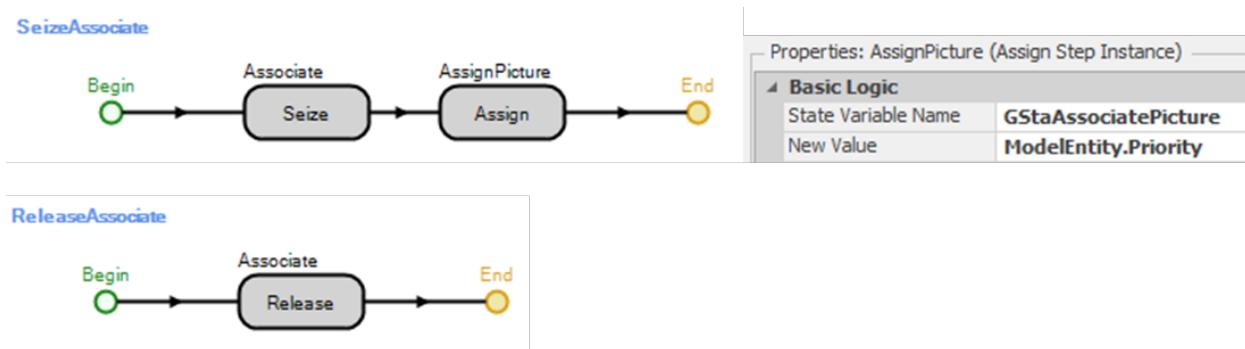


Figure 18.5: Processes for Seizing and Releasing the Associate

**Step 8:** For both the **SrvCustomer** and **SrvStorage**, specify the *Before\_Processing* add-on process trigger to be the **SeizeAssociate** process and the *After\_Processing* trigger to be the **ReleaseAssociate** in a similar fashion.<sup>276</sup>

**Step 9:** Save and run the model.

*Question 1:* When the associate finishes with a customer, does he always attend to another customer if they are waiting?

---

*Question 2:* Why does this happen, since the *Dynamic Selection* criteria of the **ResAssociate** was specified correctly?

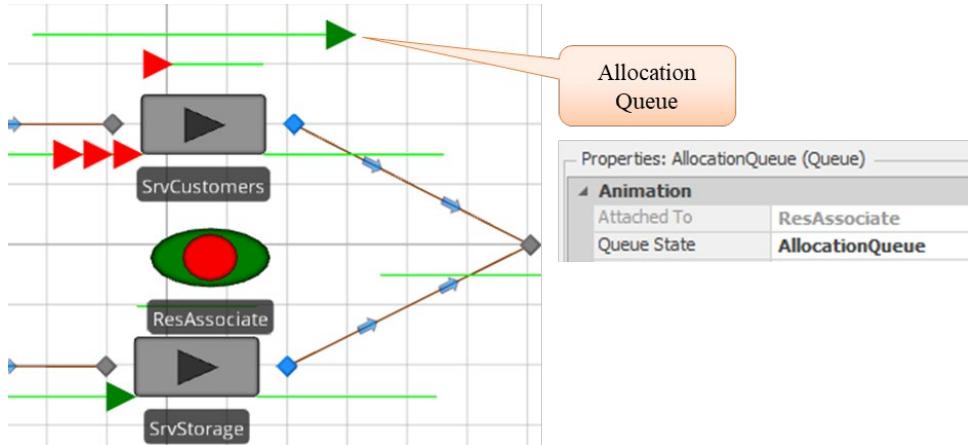
---

**Step 10:** Select the **ResAssociate**, and from the *Attached Animation* section, insert the *Allocation Queue* drawing it from left to right somewhere in the model. The allocation queue represents all the requests for the **ResAssociate** and will be ordered by priority. Save and rerun the model.

*Question 3:* What do you notice about the arrival of entities with regard to the queue?

---

<sup>276</sup> You can select both of them at the same time and specify the add-on process triggers.



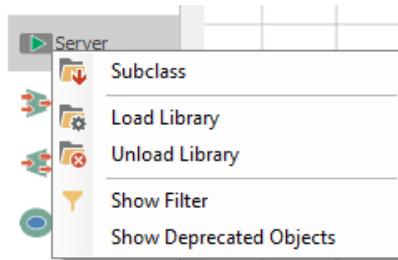
**Figure 18.6: Causing the Allocation Queue to Become Visible**

## Part 18.2: Taking an Object Apart to Figure out how it Works

For some reason, a customer waiting at the **SrvCustomer** does not register a request for service when the associate finishes with another customer. At this point, the problem may not be evident. Unlike other simulation languages, SIMIO objects can be examined by taking them apart and possibly modifying their behavior. Existing objects can be examined as well as modified in SIMIO via “subclassed.”

To try to understand why the queue has not registered the other customers, the **SERVER** will be sub-classed to understand its inner workings. Subclassing provides the ability to extend and modify an existing object.

**Step 1:** To subclass any of the standard SIMIO objects, right-mouse click on the object in the standard library and choose subclass in the menu option as seen in Figure 18.7.



**Figure 18.7: Subclassing the Server**

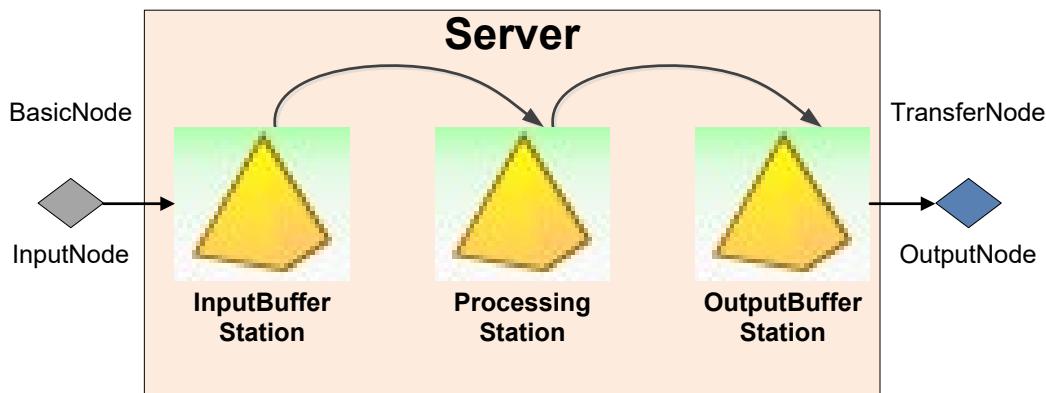
**Step 2:** After subclassing the **SERVER**, select the **MY SERVER** object in the [Navigation] panel. Now, all the characteristics of the server are visible. Look at the *Definitions→Element's* section. The **SERVER** object defines ten different elements: one **COSTCENTER**, one **FAILURE**, three **STATIONS**, one **TASKSEQUENCE**, and four **TIMER** events.

- The **COSTCENTER** element is used to define the parent activity-based costing for the object.
- The **FAILURE** element is used to define a failure mode. Failure downtime occurrences are started and ended using the *Fail* and *Repair* steps.
- The **STATION** element is used to define a capacity-constrained location within an object where one or more visiting entity objects can reside. Each **STATION** defines an **EntryQueue** and **Contents**

Queue. The **Contents Queue** contains the entities that have seized the capacity of the station and are being processed, while the **EntryQueue** will hold the entities waiting for capacity.

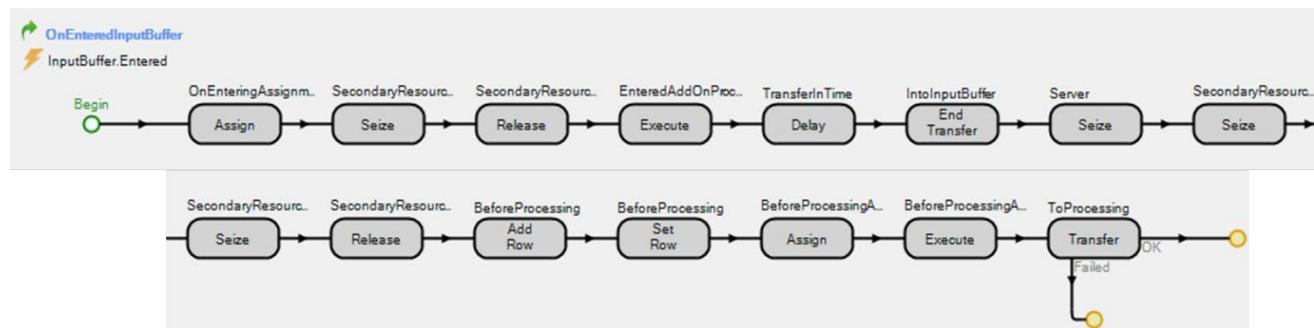
- The **TASKSEQUENCE** element is used to reference the set of tasks that compose the processing time when the *Process Type* is *Task Sequence*.
- The **TIMER** element fires a stream of events according to a specified *IntervalType* (i.e., calendar time, processing count, event count, or processing time). These timers determine when failures occur if a failure has been defined.

When entities arrive at the input node, they are automatically transferred to the “**InputBuffer**” Station if it has capacity. From the “**InputBuffer**,” the entities are transferred to the “**Processing**” station, and once they have finished processing, they will be transferred to the “**OutputBuffer**” station as seen in Figure 18.8.

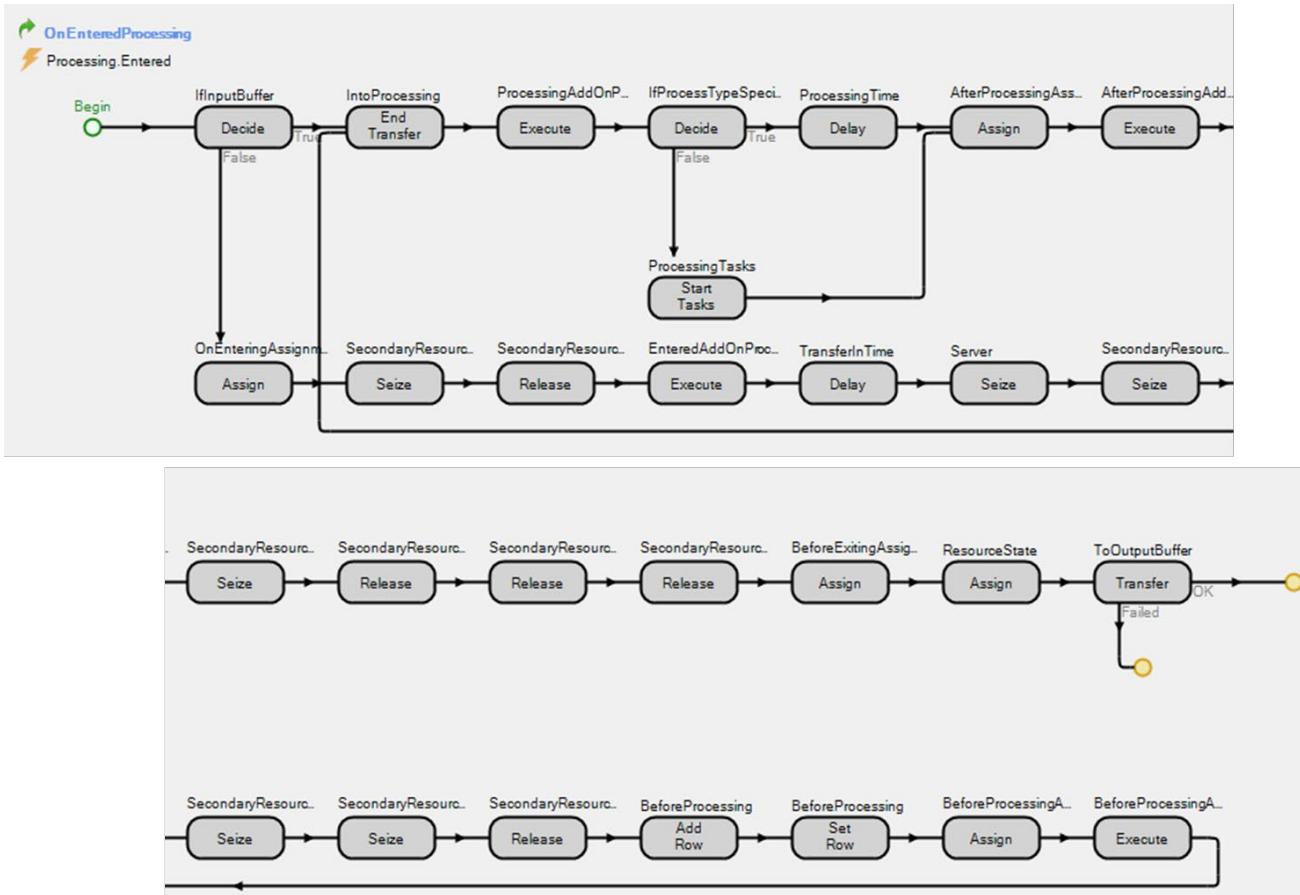


**Figure 18.8: Showing the Entity Flow through the Server Object**

**Step 3:** Next, select the “*Processes*” tab to look at the behaviors. The **SERVER** object defines 13 processes that govern how the **SERVER** reacts to failures, capacity changes, entity arrivals and exits, etc. Some of the processes respond directly to events (e.g., entities entering the Processing station, entities entering the Inputbuffer, etc.). The two processes of interest (see Figure 18.9 and Figure 18.10) are the **OnEnteredInputBuffer**, which is executed when entities enter the **InputBuffer** station from the input node if the input buffer has capacity, and the **OnEnteredProcessing** which is executed when entities enter the processing station from either the input buffer or the input node.



**Figure 18.9: OnEntered Processes for InputBuffer Station**



**Figure 18.10: OnEntered Processes for Processing Station**

When entities arrive at the SERVER, they are transferred to the **InputBuffer**, which automatically executes the **OnEnteredInputBuffer** process which has the following steps. Examination of these steps will yield some insight into our problem.

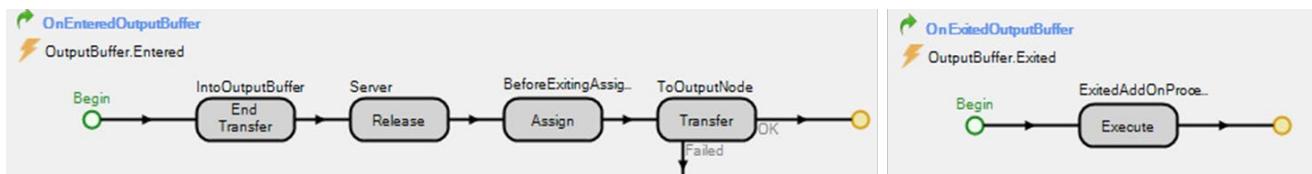
- Perform any On Entering State Assignments
- *Seize* and *Release* On Entering Secondary Resources
- *Execute* the “Entered” Add-on process trigger. The *Execute* process step will run the specified process before proceeding.
- *Delay* for the transfer time if specified.
- End the transfer of the entity into the **InputBuffer** STATION.
- *Seize* one unit of capacity of the SERVER.
- *Seize* and *Release* processing secondary Resources.
- Once the entity is able to seize capacity of the SERVER and other secondary resources, it will then execute the “*Before\_Processing*” Add-on process trigger, which in the pickup station model will seize the associate.
- Once it has finished executing the *Before\_Processing* add-on trigger, the entity is transferred to the **Processing** station, where the other process takes over.

Once entities are transferred to the processing station, the **OnEnteredProcessing** is invoked, where the actual processing is performed.

- If the input buffer capacity is zero, it goes through the same process as the `OnEnteredInputBuffer` (i.e., the false branch of the `Decide` step).
- End the transfer of the entity into the **Processing STATION**.
- *Execute* the “*Processing*” Add-on process trigger.
- Depending on “Process Type,” either start processing tasks or *Delay* the entity based on the processing time specified.
- Once the entity has been delayed or the tasks have finished, it will then execute the “*After\_Processing*” Add-on process trigger, which in the pickup station model releases the associate.
- Seize and release any Secondary Resources.
- The entity will either exit the `SERVER` or be transferred to the **OutputBuffer** station if the capacity of the output is greater than zero.

The first clue to understanding our problem is that the `Before_Processing` add-on process trigger is not executed until the entity is able to seize the capacity of the `SERVER`.

**Step 4:** Next, examine the **OutputBuffer Entered** and **Exited** processes as seen in Figure 18.11.



**Figure 18.11: OnEntered and OnExited of the Output Buffer Station.**

The `OnEnteredOutputBuffer` process will be invoked when the entity is transferred to the **OutputBuffer STATION** and goes through the following steps.

- Ends the transfer into the output buffer station.
- After ending the transfer, the entity releases capacity of the `SERVER` at this point.
- It then performs the before-exiting state assignments.
- The entities are then transferred to the output node, which invokes the `OnExitedOutputBuffer` process, which executes the “*Exited*” add-on process trigger.

**Step 5:** Now, it can be seen that the `After_Processing` add-on process trigger, which releases the associate, happens before the `SERVER` capacity is released. Therefore, the customer waiting in the input buffer cannot request the associate before the associate evaluates the current requests since it first has to seize the server before the `SeizeAssociate` process is executed.

**Step 6:** To fix the problem, set the capacity of the two `SERVERS` to “Infinity” which will eliminate the `SERVER` constraints. Therefore, every entity entering the two servers will immediately execute the `After_Processing` add-on process trigger and request the associate.

**Step 7:** Save and run the new model.

**Question 4:** After observing the model for a while, does the associate always service customers first?

**Step 8:** Setting the capacities to “Infinity” eliminates the problem. However, there are situations where the `SERVER` capacity should not be infinite because available space in the server may also limit the number of

entities being served. For instance, there may be more than one associate but no room to serve more customers. Notice from Figure 18.11 that the *Exited* add-on process trigger happens after the *Release* step and can be used to release the associate.

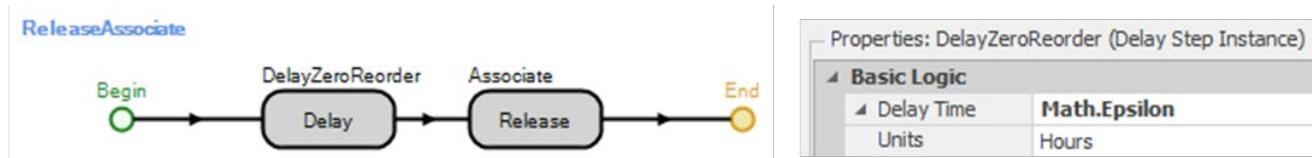
- Change the capacity of the SERVERS back to one.
- Remove the *After\_Processing* add-on process trigger from each SERVER by setting the value to null, removing it manually, or right-clicking and then selecting *Reset*.
- Choose the **ReleaseAssociate** as the *Exited* add-on process trigger for both SERVERS.

**Step 9:** Save and run the new model. You may want to increase the item arrival rate to one or two per minute.

**Question 5:** After observing the model for a while, does the associate always service customers first?

---

In other instances, having the correct order may have resulted in the same phenomena owing to zero time events, which can be checked by looking at the “Model Trace.” Recall one can reorder the zero time events by placing a *Delay* step right before a step you want to force to the end of the zero time by delaying it for zero time using `Math.Epsilon`, as seen in **Figure 18.12**. When SIMIO encounters a delay of `Math.Epsilon`, it places the steps after at the end of the current event time.

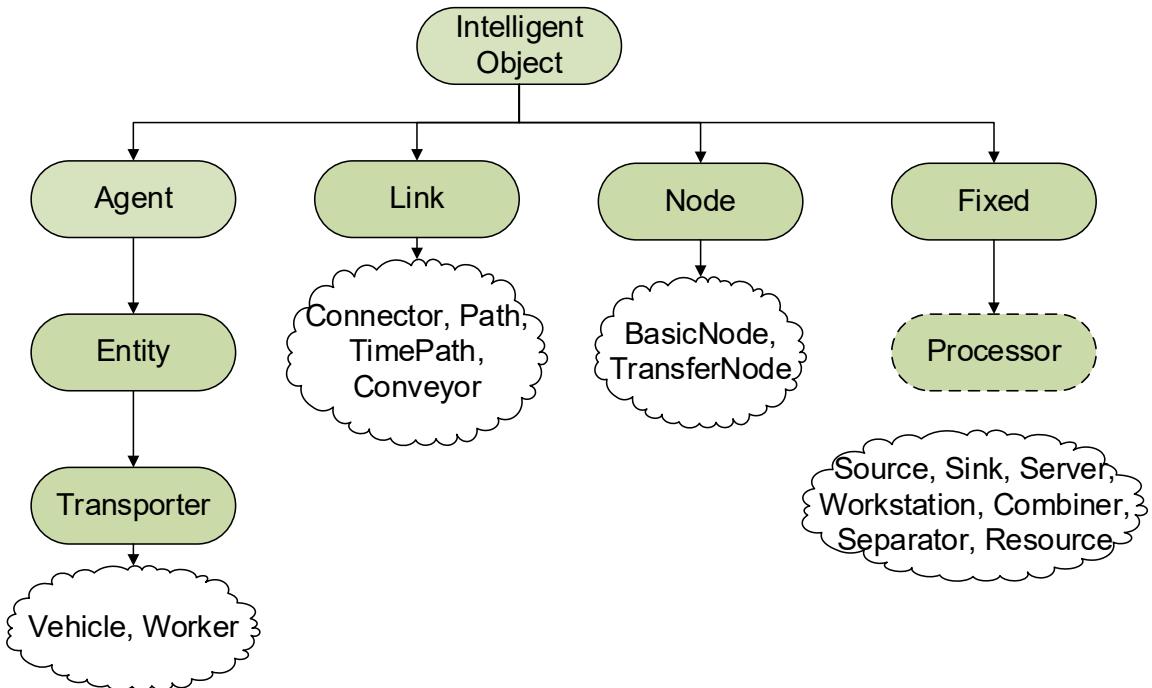


**Figure 18.12: Adding the Zero Time Delay**

### Part 18.3: SIMIO Objects and Class Hierarchy

Objects are created (instantiated) from classes. The act of clicking on a SIMIO object in the [*Standard Library*] panel and dropping it into a position on the modeling canvas causes an object or model of that type to be created (instantiated). The object will possess all the characteristics defined for that class. Once on the modeling canvas, you can change (even delete) its “Default” characteristics, such as initial property values, processing time, capacity, add-on processes, and so forth. You can also add your own properties, events, states, and elements. Each SIMIO object will have its own collection of processes, elements, properties, states, and events. Objects can be saved in a library and used in other objects. SIMIO makes no distinction between a model and an object.

Figure 18.13 shows the base object classes in SIMIO as well as the [*Standard Library*] model classes. New classes can be created from base classes, and new classes can be obtained from the [*Standard Library*] (or your library) by “sub-classing”.



**Figure 18.13: SIMIO Class Hierarchy**

The “origin” object class is the **INTELLIGENT OBJECT (IO)** class, which allows objects to be seized and released and respond to schedules. The **AGENT** class inherits the properties and characteristics of the **IO** class but adds additional characteristics and behaviors (i.e., objects that can be dynamically created and destroyed as well as move around in continuous and discrete space (grid)). The **LINK** adds the ability (i.e., properties and behaviors) for **ENTITIES** to enter, exit and move along. **ENTITIES** are specialized agents that have the ability to move on links (paths, conveyors, and connectors) as well as move in and out of **FIXED** objects (combiners, servers, workstations). **FIXED** objects represent intelligent objects that are fixed and don’t move in the space. The **TRANSPORTER** is a more specialized entity that has the additional capability of being able to pick up and drop off entities. They can do this by moving in free space or along links. Currently, models cannot be created directly from the **INTELLIGENT OBJECT** or from the **ENTITY** classes.

New object classes in SIMIO may be defined by extending or modifying the current SIMIO objects. In object-oriented terms, this method of extension is referred to as specialization or “subclassing.” When a model is subclassed, it inherits all of the characteristics (i.e., processes, elements, properties, states, and events) but has the capability to modify those inheritable features as well as add new ones.

The SIMIO [Standard Library] object classes should be viewed as model classes that have been specialized from other classes. Some of their characteristics are derived from the base classes, but some characteristics are added or changed. For example, a **WORKER** object is a specialization of an entity that can act as a **RESOURCE** and a **VEHICLE**. **WORKERS** and **VEHICLES** are specialized **TRANSPORTERS**. Several properties (e.g., **Initial Node**, **Idle Condition**, etc.) can be modified and added as well as adding new process logic to force the entity to behave similar to resources and vehicles. The **LINK**, **NODE** and **SERVER** objects have the same properties as the **FIXED** class but add more functionality. Likewise, the **PATH**, **CONNECTOR**, and **CONVEYOR** objects are more specialized versions of the **LINK** class. The **SERVER** class is a fairly general object used to constrain and delay entities in the system. The **PROCESSOR** class is a specialization of the **FIXED** class.

Extending and adding objects is an important feature of SIMIO that needs to be studied carefully. Thus, if you are only going to need the sewing machine class for one instance of one model, you may not want to take the time to create an object class. However, if you're going to use this class in multiple places, then having a reusable object will be important and its utility will be demonstrated over the next few chapters.

SIMIO uses a three-tier object hierarchy composed of the “object definition,” the “object instance,” and the “object realization.” The object definition is simply the specification of the object class through its processes, states, elements, properties, and events. The definition may be determined by the SIMIO developers for the base classes and [*Standard Library*] objects. However, a modeler may create their own object definitions (as seen in the next chapters). The object definition is shared by all instances from that class. While all objects from the same class share the same definition, each object instance has its own characteristics, such as its own add-on process, state variable value, capacity, etc. The object instance may have its characteristics changed during the simulation.

Object realizations hold the state values for the instances, and these occupy a fixed location in the model. However, ENTITIES, VEHICLES, and WORKERS are dynamic and are created from the instance definition during the execution of the simulation. So SIMIO creates static representation for that part of these object definitions that are common, but then creates a dynamic representation for the instances that move throughout the simulation. You can see this in the model with a vehicle, whose object realization shows on the modeling canvas, while the individual vehicles move around the model.

# Chapter 19

## Building New Objects via Sub-Classing: A Delay Object

In the prior chapter, we demonstrated how specializing/subclassing existing objects exposes an object's characteristics and behaviors. New characteristics and behaviors can be added to the subclassed object, or existing ones can be modified to meet specialized needs. To illustrate, we will create a new object by subclassing existing objects and changing its character. In particular, we address the need for a delay node. Sometimes, entities need to be delayed for a certain amount of time before continuing to their next destination. The SERVER object can be used to do this, but it is really more complicated than is needed to perform a simple delay. We will extend the TRANSFERNODE by adding the ability to delay entities that pass through it.

### Part 19.1: Sub-Classing the TRANSFERNODE to Create a DELAYNODE

The TRANSFERNODE is an object that is used to transfer entities out of standard objects (e.g., SERVER, SOURCE, etc.) or as transfer locations in a network of links. It provides an excellent choice to delay an entity before continuing to its next destination. The logic of riding on transporters, continuing by sequence, and taking the shortest path is already built into the object, and these properties and behaviors do not need to be reinvented in a new object. Therefore, the TRANSFERNODE will be specialized/sub-classed. This approach illustrates how an existing object is extended and modified. Objects created from existing objects are referred to as "derived" objects.

**Step 1:** First, a new project and model will be created in SIMIO.

**Step 2:** To subclass or "derive" any of the standard objects<sup>277</sup>, right-click on the object and choose subclass in the menu option, as seen in **Figure 19.1**.

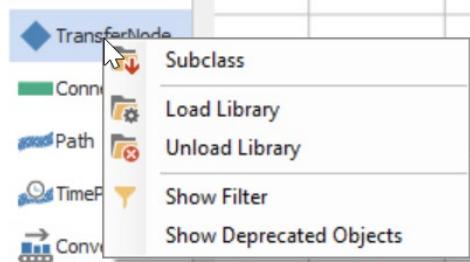


Figure 19.1: Sub-Classing the TRANSFERNODE

**Step 3:** A "derived" version of the TRANSFERNODE class is placed in the [Navigation] panel as **MYTRANSFERNODE**. Right-click the new class and select the *Model Properties* option to display all of the properties.

- Change the *Model Name* property to **DELAYTRANSFERNODE**.
- Change the *Icon* property to any appropriate icon (i.e., a PNG or bitmap file).<sup>278</sup> The icon is the picture seen in the project window used to select the object. For example, the *Delay.png* file seen in Figure 19.2 will be used in the chapter.

<sup>277</sup> You can also subclass any user created object as well.



**Figure 19.2: Icon for the DELAYTRANSFERNODE**

- Under the "General" category, change the color of the new **DELAYTRANSFERNODE** to something like maroon.<sup>279</sup>
- Select the new **DELAYTRANSFERNODE** in the [Navigation] panel, and let's take a moment to look at the inherited processes and characteristics. Navigate to the "Processes" tab and see the processes (i.e., behaviors) in **Table 19.1** that have been inherited from the parent **TRANSFERNODE** class.

**Table 19.1: Processes Inherited from the Parent TransferNode**

Process Event	Process Description
OnEntered	This process runs after the <b>leading edge</b> of an object (ENTITY, TRANSPORTER, or AGENT) has entered the node.
OnEnteredFromAssociatedObject	This process runs after an ENTITY enters the node from another object (e.g., a VEHICLE dropped it off).
OnEnteredParking	This process runs after the <b>leading edge</b> of an object (ENTITY, TRANSPORTER, or AGENT) has entered the parking station.
OnEnteredToAssociatedObject	This process runs after an ENTITY enters the node and heads to an associated object. An object uses the node as its input node (e.g., entering the SERVER object.).
OnExited	This process runs after the <b>trailing edge</b> of an object has exited the node.
OnRunEnding	This process runs when the simulation ends.
OnRunInitialized	This process runs when the simulation starts initially.
OnEnteredParking	This process runs when the ENTITY or TRANSPORTER enters the parking station, ending the transfer.
RoutingOutLogic	A process used to handle the logic of routing objects out of the node (i.e., ride on a TRANSPORTER, by sequence, etc.)
TransferFailureLogic	If an object fails to transfer to an outbound link, it will destroy the object or park it if it fails (i.e., Transporters that cannot be destroyed).

- Select the "Definitions" tab to access the characteristics of the node. Elements represent objects in a process that change state over time (i.e., statistics, timers, monitors, failures, etc.). There are two elements defined: a parking STATION and a ROUTING GROUP.
  - The PARKINGSTATION station is used to house entities, operators, and transporters parked at the node. The station element will be discussed in more detail in the next chapter.
  - The ROUTINGGROUP element is used with a *Route* step to route an entity object to a destination selected from a list of candidate nodes. The TRANSFERNODE object uses a RoutingGroup element in its internal logic to determine where the entities will travel when they leave the node.

<sup>278</sup> Icons are typically 32 pixels wide by 32 pixels high. Many image editing software can create these icons (e.g., Photoshop, Paint or freeware GIMP program. (<http://www.gimp.org>)).

<sup>279</sup> You cannot change the picture (i.e., diamond) because nodes are special objects that take up zero space in the actual system.

- The new sub-classed node inherits several properties and states and four events from the parent TRANSERNODE, as described in Table 19.2.<sup>280</sup>

**Table 19.2: Events Inherited from the Parent TransferNode**

Event	Event Description
CapacityChanged	The node's capacity has changed.
Entered	An object's leading edge has entered the node.
Exited	An object's trailing edge has left the node.
RiderWaiting	An ENTITY is waiting for transport at this node.

- Three string lists (i.e., **EntityDestinationType**, **RideOnTransporterType**, and **ActionConditionType**) have been defined and used in property dropdowns. The **EntityDestinationType** list is used to specify the property of how to route the entity to its next destination node (i.e., Continue, Specific, BySequence, SelectList, UseCustomRoutingGroup, and Destroy Entity). The **RideOnTransporterType** list allows the user to specify whether the entity must ride on a **TRANSPORTER** to exit the **TRANSERNODE**. In contrast, the **ActionConditionType** is the condition type (NoCondition, IsEntity, IsTransporter, CustomCondition) used to compute Tally statistics at this node.

## Part 19.2: Modifying Processes and Adding Properties for the New Node

The new delay node sub-classed from the standard **TRANSERNODE** will operate like the original node since nothing has been added or modified. The node needs to delay the objects that enter the node either individually or via a transporter.

**Step 1:** Under the "Definitions" tab in the delay node, add an *Expression* property<sup>281</sup> named **DelayTime**, which a user can use to specify the delay time for the entity. Specify the *Default Value* property to be zero while the *Unit Type* should be "Time" with *Default Units* of "minutes," as seen in **Figure 19.3**. Place the new property underneath the "Process Logic" category.

---

<sup>280</sup> To access Inherited items, click on the  to make the items visible.

<sup>281</sup> The *Expression* Property allows the user to specify any mathematical expression similar to a *Processing Time* of the SERVER.

Properties: DelayTime (Expression Property)	
Value	
Default Value	0.0
Switch Property Name	
Switch Condition	Equal
Switch Value	
Candidate References	False
Unit Type	Time
Default Units	Minutes
Appearance	
Display Name	DelayTime
Category Name	Process Logic
Expanded	False
Parent Property Name	
General	
Name	DelayTime

Figure 19.3: Inserting a Delay Expression Property

Every time an entity enters this transfer node either on its own or via an associated object, it should be delayed by the delay time expression property. Therefore, the OnEnteredFromAssociatedObject<sup>282</sup> and the OnEntered processes will need to be modified to reflect this consideration. Currently, the processes are not editable, which can be seen by selecting the OnEntered process and noticing the steps and properties are grayed out. After selecting the process,<sup>283</sup> click the *Override* button, allowing changes to be made as seen in . Once the process has been overridden, the *Override* symbol is placed under the process name to indicate it has been modified. Note: The Restore button can be used to return the process to its original state.

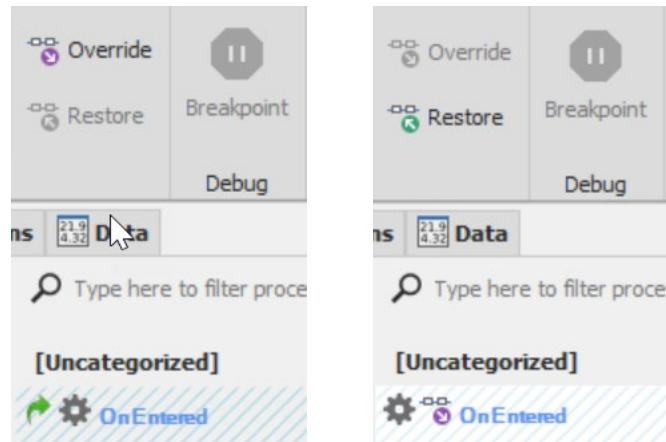


Figure 19.4: Overriding and Restoring Inherited Processes

The OnEntered process is relatively complex and has the following basic steps.

- *On Entering State Assignments* are performed.
- On Entering Tally statistics to be collected if specified.
- *Fires the "Entered" Event.*
- Then *Execute* the "EnteredAddOnProcess" that a user may have specified.

<sup>282</sup> For example, the object is entering via transporter.

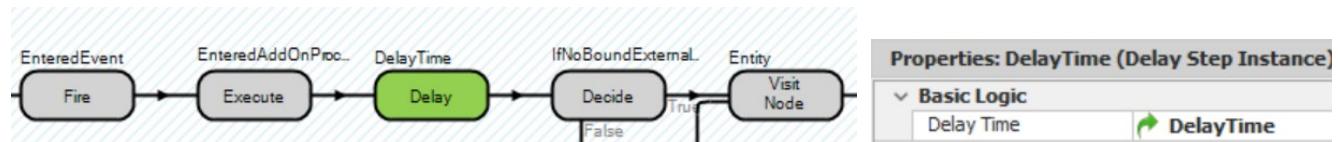
<sup>283</sup> Select the *Processes* tab to access all the processes.

- Checks to see if a node is "Bound to an External Node" and, if so, transfers it to the specified parent external node.
- The *Visit Node* step is used to invoke the entity's `OnVisitingNode` process to indicate the entity has arrived at the destination node.
- If this entity is a transporter, transfer it to the outbound link since it does not go through the same logic, and if there is no outbound link, the transporter is sent to free space or processes a transfer error.
- If the entity is not a transporter, then it executes the `RoutingOutLogic` process to transfer the entity out of the node with the correct logic (i.e., uses the "Route" step in conjunction with the `ROUTINGGROUP` element). If not riding on a transporter in routing out, then transfer to the outbound link occurs, and the prior logic applies.

*Question 1:* Where should the delay step be inserted into the process?

---

**Step 2:** Users may sometimes want to change the delay time based on some condition when they enter the node. Therefore, insert a *Delay* step after the "*Entered*" *Add-On Process* is executed to allow the processing time to be potentially set in the add-on process before the delay occurs. The *Delay Time* property should be set to the **DelayTime** reference set in the previous step, as seen in Figure 19.5.<sup>284</sup> Under the "General" Section, the *Name* property can be specified as "**DelayTime**" for a better description.



**Figure 19.5: Modifying the OnEntered Process to Include the Delay**

**Step 3:** Repeat the procedure for the `OnEnteredFromAssociatedObject` process by adding the *Delay* step in the same location. Once the process has been overridden, copy the delay step and paste it into the appropriate place in this process.

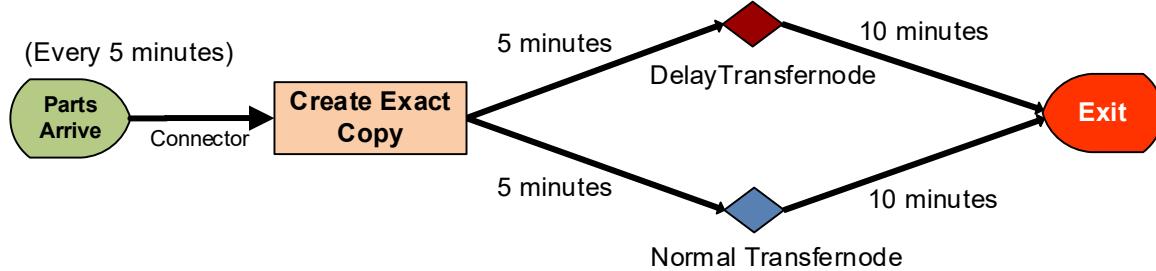
**Step 4:** We may want to add the delay to the `OnEnteredToAssociatedObject` if you were to use it as an external node for another object. Just override the process and copy the *Delay* step between the execution of the `Entered_AddOnProcess` and the decision on whether the node is bound to an external node.

### Part 19.3: Creating a Model to Test the New **DELAYTRANSFERNODE**

Now that the new delay transfer node has been created, it can be used in any model where the original `TRANSFERNODE` is used with the added benefit of delaying entities for some time as they enter the node. Figure 19.6 shows a network where we will test our new **DELAYTRANSFERNODE**.

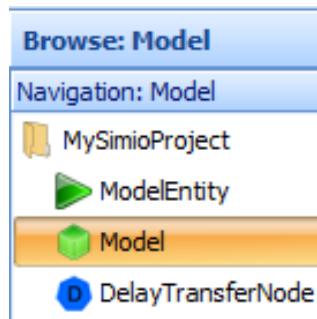
---

<sup>284</sup> Right click on the property label to specify a referenced property.



**Figure 19.6: Building an Example to Test Our DelayTransfertNode**

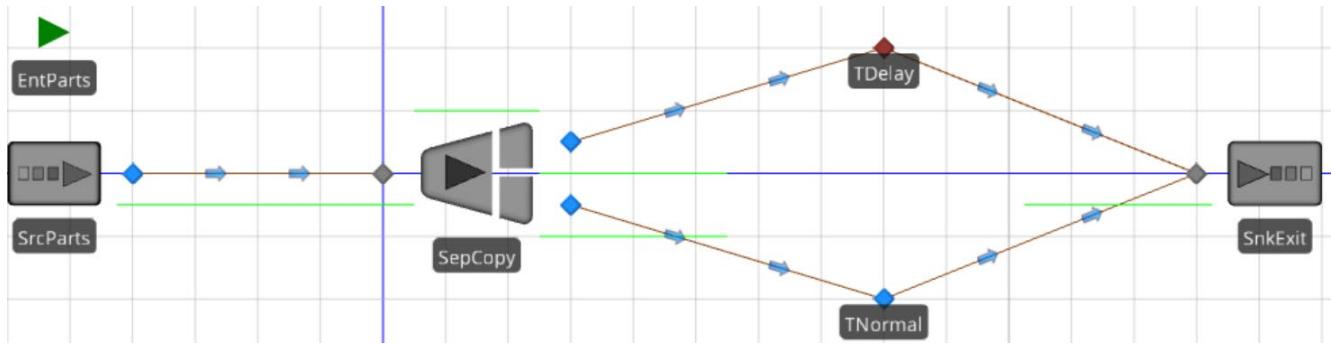
**Step 1:** Go back to the original Model by selecting it in the Navigation panel.



**Figure 19.7: Navigating Back to the Original Fixed Model**

As seen in Figure 19.8, the model will create an entity and then duplicate it, sending the original through a **DELAYTRANSFERNODE** and the duplicate through a standard **TRANSFERNODE**. The time in the system will be used to illustrate the delaying behavior.

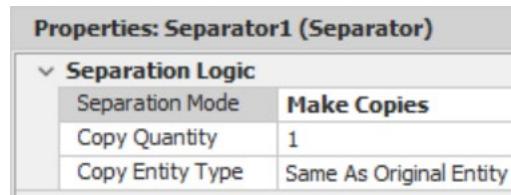
- Insert a new **MODELENTITY** named **EntParts**.
- Insert a **SOURCE** named **SrcParts** that creates **EntParts**.
- Add a **SEPARATOR** named **SepCopy**, which can be used to create copies of entities or to un-batch entities that have been combined/batched using a **Combiner**.
- Add a new **DELAYTRANSFERNODE** named **TDelay**, which can be accessed via the **[Project Library]** panel.
- Add a normal **TRANSFERNODE** named **TNormal**
- Add a sink named **SnkExit**.
- Connect the **SrcParts** to the **SEPARATOR** using a connector.
- Connect the parent output to your new **TDelay** and the member output to the **TNormal** node via **TIMEPATHS**, which takes five minutes for each.
- Connect both **TNormal** and **TDelay** to the **SINK** via ten-minute time paths.



**Figure 19.8: Model Used to Demonstrate the New Delay TransferNode**

**Step 2:** Configure all the fixed objects with the following parameters.

- SOURCE: Parts should arrive at a constant interarrival time of five minutes.
- SEPARATOR: The *Separation Mode* property should be specified as "Make Copies," and the *Copy Quantity* property should be one, as seen in Figure 19.9. This will make one copy of the current entity. All properties of the entity will be copied. However, the entity's creation time is current and not necessarily the same time as the original entity. The original entity will flow out of the **ParentOutput** node, while the copy will flow out of the **MemberOutput** node.



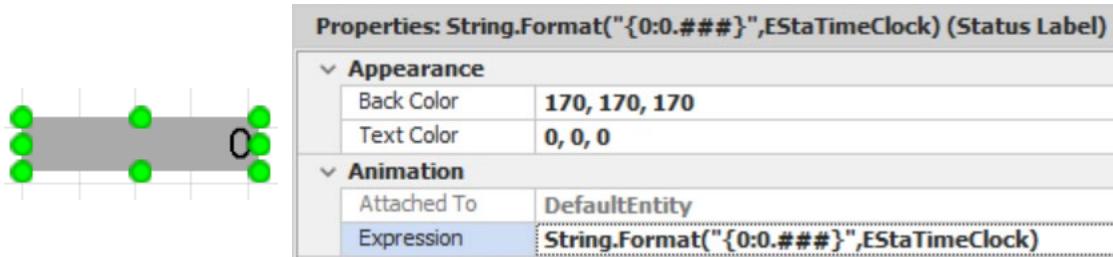
**Figure 19.9: Separator Properties**

**Step 3:** A new characteristic needs to be added to the **EntParts** MODEL ENTITY to track the current time at specific points in the system. Select the **ModelEntity** in the [Navigation] panel and add a new Discrete Real State variable named **EStaTimeClock** underneath the "*Definitions*" tab. A real state variable is needed since the value will change throughout the run of the system. Make the *Unit Type* **Time** and the *Units* **Minutes**.

**Step 4:** Let's add a status label to the ENTITY that displays its associated **EStaTimeClock** state variable. Select the **EntParts** object in your model and then choose the *Status Label* under the *Attached Animation*→*Dynamic Text* section.<sup>285</sup> Draw a slender status label near the entity with the expression equal to `String.Format("{0:0.###}", EStaTimeClock)`<sup>286</sup>, as seen in **Figure 19.10**.

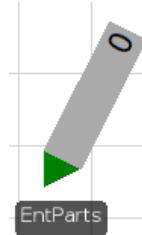
<sup>285</sup> We could put the expression in the *Dynamic Label Text* under the *Animation* section of the entity as well but this is more visible.

<sup>286</sup> This will format the time clock to have three digits of precision.



**Figure 19.10: Creating a Status Label to Track the Part's EStaTimeClock State**

**Step 5:** Since the status label is attached to the **MODEL ENTITY**, it will travel with it. Therefore, rotate (via *Ctrl→Mouse*), size, and move the label so it sits right on top of the entity, as seen in Figure 19.11 below.



**Figure 19.11: Attaching the Status Label to the Part Entity**

**Step 6:** Save and run the model to make sure it behaves as predicted.

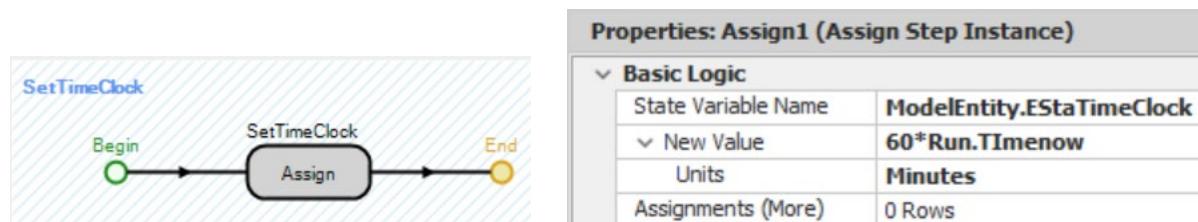
**Question 2:** Does the part get duplicated at the separator correctly?

---

**Question 3:** Does the status label travel with the **EntParts**, and what is the value of all the labels?

---

**Step 7:** The value of the **EStaTimeClock** state variable has not been updated at this point. Navigate to the "Processes" tab and create a new process named **SetTimeClock**. Insert an *Assign* step that sets the **ModelError.EStaTimeClock** to a new value of  $60 * \text{Run.TimeNow}$ <sup>287</sup>, which will set the variable to the current simulation time in minutes.



**Figure 19.12: Creating a Generic Process to Set the Model Entity's StaTimeClock**

**Step 8:** When the **Parts** exit the **SEPARATOR** through the **Parent** or **Member** output, the **Parts EStaTimeClock** variable needs to be set to the current time. Select the **ParentOutput@SepCopy** transfer

---

<sup>287</sup> `Run.TimeNow` is a function that returns the current time of the simulation in hours.

node and specify the **SetTimeClock** process for the "*Entered*" add-on process trigger, which will assign the current simulation time to the **EntPart's EStaTimeClock**. Perform the same process for the **MemberOutput@SepCopy "Entered"** add-on process trigger.<sup>288</sup>

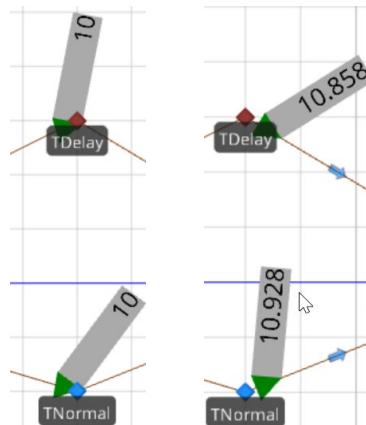
**Step 9:** Repeat the process for the "*Entered*" and "*Exited*" add-on process triggers for both the **TNormal** and **TDelay** transfer nodes.

**Step 10:** Save and run the model. Observe the time stamps as the entities leave the separator as well as the times to enter and exit the transfer nodes. You would think that the time stamps should be the same since the delay time is zero and whole numbers are used for the arrival and time paths. You may want to set a breakpoint on one of the nodes and then a single step until the entity leaves the transfer node.

*Question 4:* Are the entering and exiting times different for the parts, and if so, why?

---

It seems there is a small amount of time being delayed in one of the transfer nodes compared to one other. They enter the nodes at the same time (i.e., at time ten on the left side of Figure 19.13) but appear to leave at slightly different times, as seen on the right side of Figure 19.13. However, transfer nodes take up zero space and should not have caused any delay (i.e., discrepancy). The time an entity enters the node should equal the time it leaves.

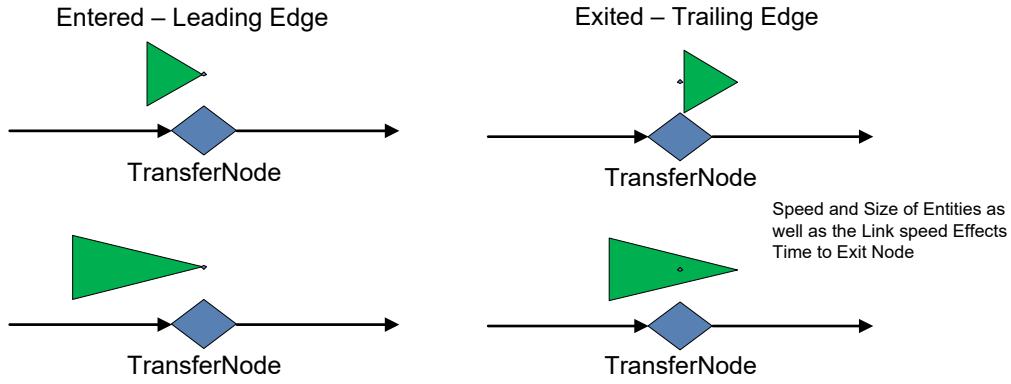


**Figure 19.13: Showing Discrepancies Between Exited and Entered Times**

The discrepancy deals with when the "*Exited*" process event is fired. This event will not fire until the trailing edge of the entity has left the node, which depends on the length of the entity and how fast the entity is transferred onto the link (i.e., a function of the entity's speed and the speed of the path its leaving and the one it is entering). The "*Entered*" process event is fired once the leading edge has entered the node. Therefore, when setting time stamps, the selected add-on process trigger should be based on the leading edge of an ENTITY. Figure 19.14 demonstrates that both entities enter the node at the same time. However, the smaller entity will exit the node first, as shown.

---

<sup>288</sup> You can select them all and specify the "*Entered*" process at one-time.



**Figure 19.14: Demonstrating Entering and Exiting from Nodes**

**Step 11:** Reset (remove) the "Exited" add-on processes of both transfer nodes (i.e., **TDelay** and **TNormal**) and reset the "Entered" two outputs of the **SepCopy** to remove the add-on process trigger.<sup>289</sup> Specify the **SetTimeClock** process for the "Entered" add-on process for each of the four-time paths leaving the two transfer nodes and the SEPARATOR.

**Step 12:** Save and run the model to demonstrate that the entities are correctly synced.

**Question 5:** Are the time stamps whole numbers and synced?

---

**Step 13:** Next, specify a delay time of four minutes for the **TDelay** transfer node and rerun the model. The part entities are now being delayed four minutes before proceeding into the rest of the network.

**Step 14:** Change the arrival rate to every two minutes and observe what happens.

**Question 6:** What happens to multiple entities that arrive at the delay node?

---

#### Part 19.4: Commentary

- Derived objects are created by subclassing in SIMIO. Derived objects add or change characteristics and behaviors of existing objects. This chapter added delay features to the transfer node and the processing was modified accordingly.

---

<sup>289</sup> Recall to reset a process trigger, right click the process and select the “Reset” item.

# Chapter 20 Creating New Objects

---

Traditionally, an object is created (instantiated) from an object class. The class describes an object's composition in terms of its attributes and the methods (procedures) it performs. In SIMIO, the model itself is an object (from the `FIXED CLASS`). When SIMIO is initiated, a `MODEL` and a `MODELENTITY` are automatically created within a new `PROJECT LIBRARY`. The `MODELENTITY` defines the `DEFAULTENTITY` as part of the project, and instances of this run-time class can be created throughout the simulation run. The `MODEL` is formed by combining instances/objects from the libraries available, usually including the [*Standard Library*]. It is not possible in SIMIO to create utterly independent simulation objects. Models must be formed from either the base classes or by “subclassed” an existing object class. The five base classes are `FIXED CLASS`, `ENTITY CLASS`, `TRANSPORTER CLASS`, `NODE CLASS`, and `LINK CLASS`. There is a `PROCESSOR` class derived from the `FIXED CLASS` to help create processing objects.

When developing new objects, the user must adopt an “architect’s view” and a “user’s view” since few object classes are developed to create only a single object. Instead, new object classes, like a `SERVER` or `TRANSFER NODE` in SIMIO, are expected to be used many times in one or more simulation projects. Therefore, the purpose or scope of an object’s use becomes central to its design. SIMIO object classes have been developed for many purposes and with diverse users in mind.

Consider the `SERVER` object class, which is basically designed as a single queue (input buffer) with fixed or scheduled multiple identical servers (at processing). However, it allows for many specifications, including static and dynamic ranking, transfer-in time, flexible processing time with the option of tasks, buffer capacities, material handling, reliability of servers, activity-based accounting, secondary resources, state assignments, etc. In other words, this object class is designed for a wide variety of uses. While state assignments, the acquisition of additional resources, and many of the additional features could be user-implemented in SIMIO processes, they are made “convenient” specifications for users not wanting to write processes, making the object class more accessible to casual or learning users. As you review the design of the `SERVER`, you can begin to understand the architectural “paradox.” Namely, you want your objects to be “powerful” with many features for widespread application, yet you want them to be “simple” so they can be quickly understood and used. It’s not an easy compromise. Generally, the SIMIO objects tend to favor the powerful side of the paradox, while in our development, we will tend to explore the simple side.

Object classes with many features like those in SIMIO are typically called “heavyweight” because each object requires a lot of information and has numerous procedures.<sup>290</sup> Lightweight objects may have limited use but are easy to understand and use and, if appropriately designed, can become easy to extend through object inheritance and/or composition. This chapter will focus on creating lightweight specialized objects, objects with limited purpose (compared to SIMIO objects). However, in these creations, significant references are made to the SIMIO objects as guides for development, and the new object classes will become a part of our custom library. Also, we use the words “object” and “model” interchangeably, which SIMIO also uses.

## Part 20.1: Creating a Simple Delay Object

The previous chapter created a new delay transfer object by subclassing the standard `TRANSFERNODE` to delay entities that enter the node. However, the entities will queue up at the transfer node right on top of one another. Also, there is no opportunity to add a different symbol (i.e., an oven, etc.) to represent the delay location since the transfer node takes up zero space. A new fixed object (i.e., a simple delay object) will be built that only delays an entity in its path, similar to any of the standard objects. Of course, such a delay can be accomplished by an infinite capacity `SERVER` or even just a `TIMEPATH`, but creating this simple delay object will be instructive both in its creation and as a base for extension.

---

<sup>290</sup> The Vehicle/Worker is clearly the most complicated of the SIMIO standard object classes.

**Step 1:** Load the model from the previous chapter and save it as Chapter20-1.spfx.

**Step 2:** Insert a “New Model” icon, select the “Fixed Class” option to produce a new object class type, and name it **SimpleDelay**. In the “Advanced Options” properties of this object class, be sure that its *Resource Object* property is “*False*” since we don’t plan to use these objects as resources.<sup>291</sup>

**Step 3:** Within this new object class, add a new expression property named **DelayTime**. Its *Default Value* is `Random.Pert(1, 2, 4)`.<sup>292</sup> The *Unit Type* would be *Time*, and the *Default Units* should be *Minutes*. Finally, specify “*Process Logic*” as the *Category Name*.

**Step 4:** Objects that contain or constrain entities (SOURCES, SERVERS, COMBINERS, SEPARATORS, WORKSTATIONS, etc.) utilize **STATION** elements to house these entities. The **STATION** element may be used to define a capacity-constrained location within an object where one or more visiting entity objects can reside. From the *Definitions*→*Element*, insert a new **STATION** named **StationDelay**. Its *Initial Capacity* should be *Infinity* because we don’t want to restrict the number of entities using the station.

The station element has two queues associated with each station: **Contents** and **EntryQueue**. The **EntryQueue** is the queue where entities wait until there is available capacity to allow the entities to move into the **Contents** queue. Suppose you specify that the input buffer is ten for a SERVER, and the eleventh entity arrives at the server. In that case, the entity will be placed into the **EntryQueue** and not be allowed to enter the server (i.e., it will stay at the input node of the server).

Notice the `Server.InputBuffer.Contents` queue has already been used to determine the number of entities waiting in the input buffer to be processed or the `Server.Processing.Contents` are used to determine the number of entities that are currently being processed at a SERVER.

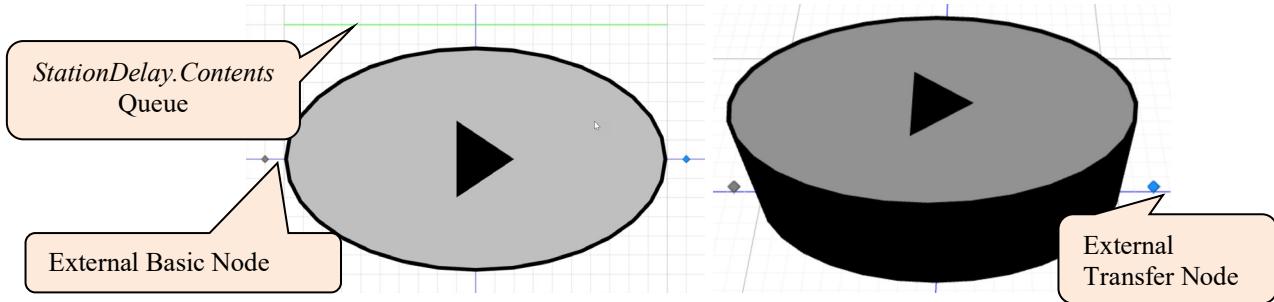
**Step 5:** The delay **STATION** element allows the entity to reside while it is being delayed. However, entities need a way to enter and exit the **DelayObject**. The external view creates the visual interface and provides enter and exiting nodes to the object. Go to the “External” section underneath the *Definitions* tab to create the visual interface.

- First, we need to define the “symbol” to represent objects from this class. You can use a symbol from the existing SIMIO symbol library, create your own using the *Drawing* tab, download a symbol from the Trimble 3D Warehouse, or import from several CAD programs (e.g., SketchUp © 2023). In this case, we will create a symbol using the drawing shown in Figure 20.1.
- From the *Drawing* section, insert an Ellipse centering it on the grid. In the *Object* section, change the *Object Height* to 0.75 and the *Line Width* to 0.02. From the 3D view, color the top grey and the side and line black. Make the drawing roughly two meters wide.
- Next, insert a half diamond to represent the delay. Using the *Polygon* tool, just define the top, right, and bottom points. Place the shape so it is positioned halfway on the ellipse. Switch into 3-D view, select the shape, and raise the label while holding the Shift key down so that it’s barely above the ellipse. Switch back to the 2-D view to center in Figure 20.1.

---

<sup>291</sup> Most SIMIO object are resource bound and have capacity.

<sup>292</sup> Since the expression editor is not active, type the expression directly into the *Default Value*. We will tend to use the random number stream specification throughout to aid comparisons.



**Figure 20.1: Delay Symbol**

**Step 6:** Add the **Input** and **Output** nodes, which provide the mechanism for objects to arrive and leave the **DELAY** object. Click the *External Node* from the *Drawing→Transfers* section to add two external nodes and position them to the edges of the object, as seen in Figure 20.1. External nodes have specifications explained in Table 20.1 that need to be defined.

**Table 20.1: External Node Specifications**

Specification	Description
<i>Node Class Name</i>	The type of node this external node should be ( <b>BASICNODE</b> , <b>TRANSFERNODE</b> , <b>DELAYTRANSFERNODE</b> , etc.). Transfer nodes are typically used as exit/output nodes since they contain the logic to route entities to the next destination, while basic nodes are used as entrance/input nodes.
<i>Input Location Type</i>	Location type in the object that an entity transfers into using this external node (e.g., Station, Node, or Container). If you have added objects to the facility window and want to transfer the entities to one of the nodes in the fixed model of the new object, select “Node.”
<i>Station Name</i>	Which station will the entity transfer from this node if “Station” is the input location type?
<i>Node Name</i>	Which node in the facility of the object should entities be transferred from this external node if “Node” is the Input Location Type?
<i>Container Name</i>	Which container in the object’s facility should entities be transferred from this external node if the <i>Input Location Type</i> is “Container.”

- The **Input** node should be a **BASICNODE** as specified in Figure 20.2. When entities enter the node, we want SIMIO to automatically transfer them to the **STATION StationDelay**.
- The **Output** node will be a **TRANSFERNODE** to allow entities to be routed and/or ride on a transporter.

<b>Properties: Input (ExternalNode Instance)</b> <div style="border: 1px solid #ccc; padding: 5px;"> <b>Advanced Options</b>            Initial Property Values   0 Rows  <b>Basic Logic</b>            Node Class Name   <b>BasicNode</b>            Input Location Type   <b>Station</b>            Station Name   <b>StationDelay</b>  <b>General</b>            Name   <b>Input</b> </div>	<b>Properties: Output (ExternalNode Instance)</b> <div style="border: 1px solid #ccc; padding: 5px;"> <b>Advanced Options</b>            Initial Property Values   0 Rows  <b>Basic Logic</b>            Node Class Name   <b>TransferNode</b>            Input Location Type   None  <b>General</b>            Name   <b>Output</b>            Description         </div>
--	--

**Figure 20.2: Specification of the External Nodes**

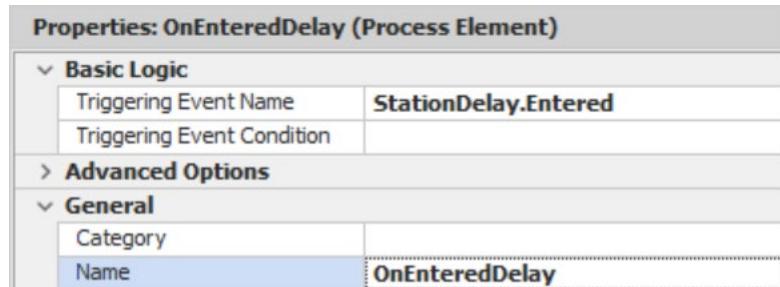
**Step 7:** Unlike the **DelayTransferNode**, the entities are contained in the station in one of the two queues. As entities enter the delay object and are delayed, the entities can be visually seen in the queue. Insert the animated queue above the symbol by selecting the *Queue* under the “Animation” tab. The *QueueState* should be the *Contents Queue* of the **StationDelay** (i.e., **StationDelay.Contents**).

**Step 8:** Since we have defined the default external view, the logic to process the entities entering the objects needs to be specified. Stations automatically respond to three events, as described in Table 20.2.

**Table 20.2: Events at a Station**

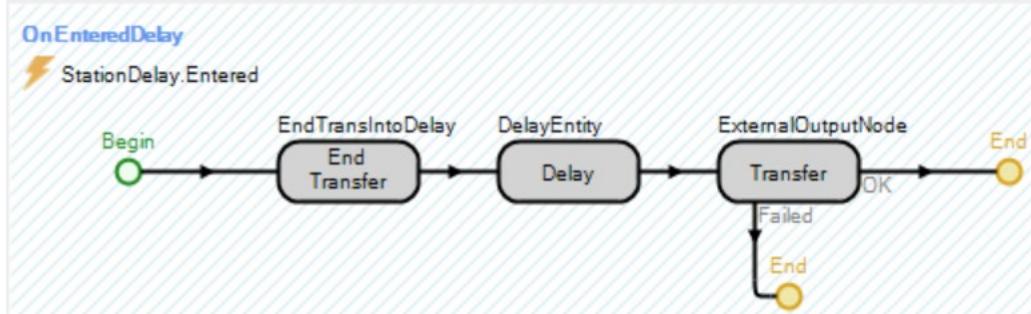
Event	Description
Entered	Fired when entities leading edge enters the station
Exited	Fired when entities exit the station
CapacityChanged	Fired when the capacity of the station changes

- Create a new process named **OnEnteredDelay**, which will be executed every time an entity enters the STATION by specifying the *Triggering Event* as `StationDelay.Entered` as seen in Figure 20.3.



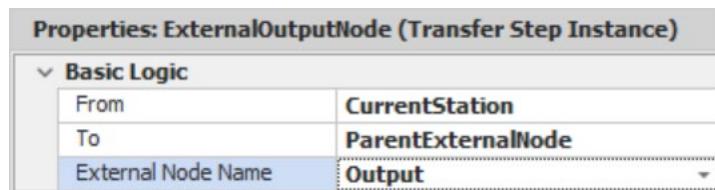
**Figure 20.3: Specifying the Entered Triggering Event of the Delay Process**

- The first thing that needs to happen is that the entity has to end the transfer into the station using an *End Transfer* step, which indicates that the entity has finished the transfer. This step also fires the entity's transferred event, which tells the input node that the process has been completed.



**Figure 20.4: Station Activity**

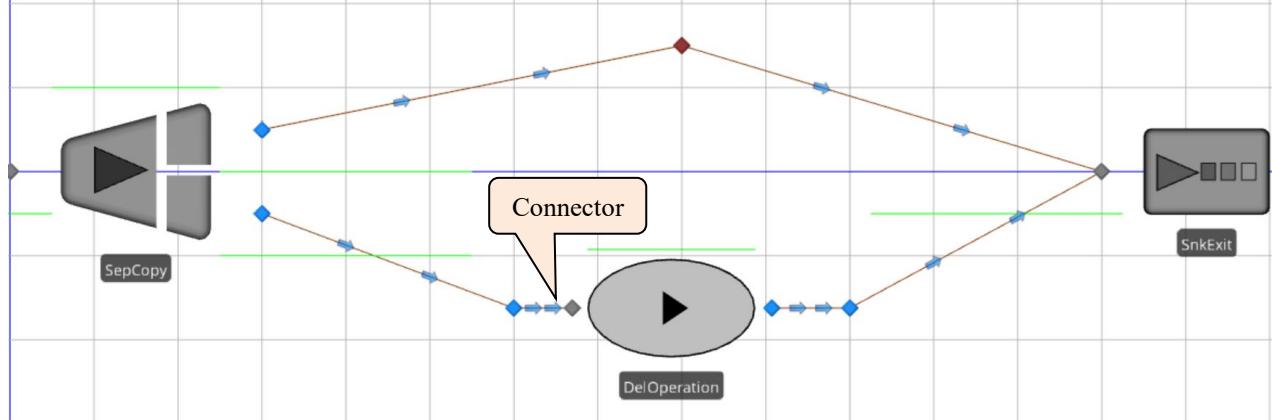
- Next, add a Delay step similar to before (see Chapter 19) specifying the **DelayTime** as a referenced property.
- Once the entity has been delayed, it needs to be transferred out of the **SIMPLEDELAY** using a *Transfer* step as specified in Figure 20.5. Notice the entity is sent to the external output node to enter back into the parent model when it leaves the new object:



**Figure 20.5: Transfer Step Specifications**

**Step 9:** Next, we need to use the new DelayObject in our model, as seen in Figure 20.6. Switch back to the main create **Model** (within the *Navigation* window). We will insert the delay between two nodes to facilitate multiple changes as we modify the object to avoid setting the paths each time.

- Delete the **TIMEPATH** connecting the **TNormal** to the **SnkExit**. Insert a new **TRANSFERNODE** and connect it to the **SnkExit** via a ten minute **TIMEPATH**. Also, you will need to specify the **SetTimeClock** as the *Entered* add-on process trigger.
- Insert a **SimpleDelay** object from the [*Project Library*] and call it **DelOperation**. Specify the delay time to be four minutes to match the **TDelay** time. Use connectors to link the input and output nodes.



**Figure 20.6: SIMO Model with Simple Delay**

**Step 10:** Save and run the model for 24 hours, observing the differences between the delay object and the delay transfer node. Set the arrival rate to two minutes to see the differences. Note, if your entities are upside down in the queue, then the animated queue in the “*External*” view was drawn in the wrong direction (i.e., it should be right to left).

**Question 1:** What is the visual difference between the **DELAYTRANSFERNODE** and the **DELAYOBJECT** concerning the flow of entities?

---

## Part 20.2: Adding Color and Processing Contents

One of the features of the SIMIO objects (e.g., **SERVER**) is they change color based on their status (state). With some work, we can mimic this behavior in our delay object, so it turns a different color when entities are being delayed versus when the delay object is idle.

**Step 1:** The **SERVER** keeps track of its states using a **LIST STATE** variable named “**ResourceState**.” The **LIST STATE VARIABLE** defines a discrete integer variable with a list of possible values from zero to  $N$ . You need first to define a string list with the names. Select the **DelayObject** in the Navigation panel. From the *Definitions*→*Lists*, insert a new string list named **LstDelayStateNames** with “Idle” and “Delaying” as the two potential states.

Name	
▼ <b>Strings</b>	
<b>LstDelayStateNames</b>	
	String
0	<b>Idle</b>
1	<b>Delaying</b>

**Figure 20.7: Specifying the Names of the Potential States**

**Step 2:** Insert a new LIST STATE variable named **StaDelayState**, which uses the **LstDelayStateNames** list. Since SIMIO will automatically track statistics on each state, specify the *Data Source* and *Category* property as seen in Figure 20.8.

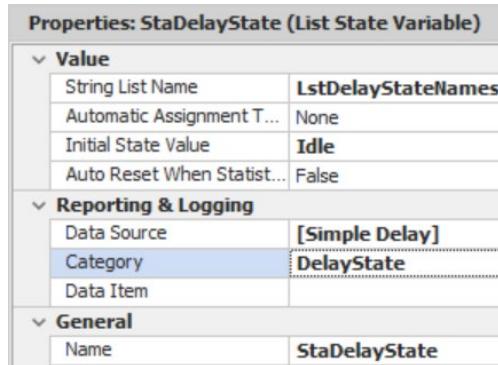


Figure 20.8: Setting up the List State Variable.

**Step 3:** Once the entity enters the delay station, the delay state will be “Delaying” and will potentially become idle when it leaves the simple delay object, as seen in Figure 20.9.

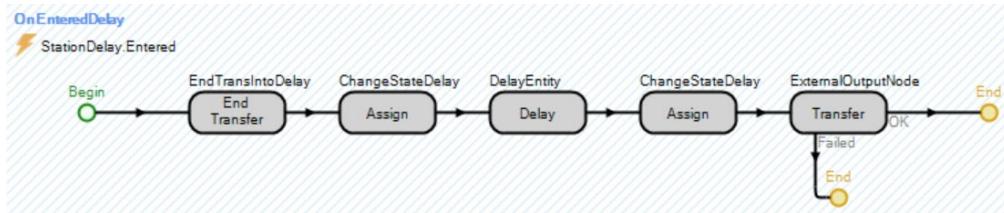


Figure 20.9: Modifying the **OnEnteredDelay** to Change the Delay State

- Insert an *Assign* step that sets the **StaDelayState** to “1” immediately after the *End Transfer*.
- Right before the entity leaves (i.e., the *Transfer* step), we need to update the status to idle if this is the only entity in the delay station. Insert another *Assign* step that sets the **StaDelayState** variable to **StationDelay.Contents.NumberWaiting>1**, which will evaluate to zero if this is the last entity in the delay station or one, as seen in Figure 20.10.

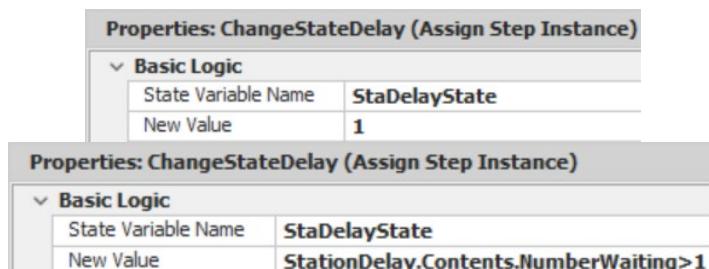


Figure 20.10: Changing the State of the SIMPLEDELAY Object

**Step 4:** Return to the main **Model** and set the source’s arrival time to **Exponential(5)** minutes.

**Step 5:** Save and run the model for 24 hours, looking at the results.

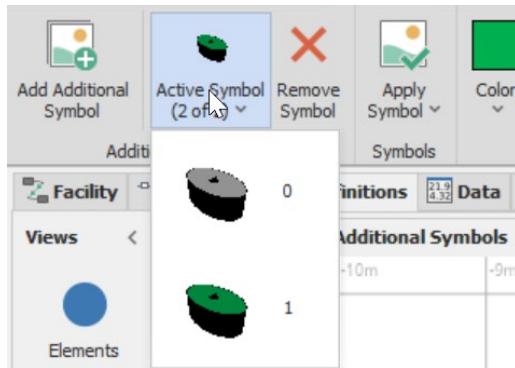
**Question 2:** What percentage of the time was the delay object idle and delaying?

---

**Question 3:** While the simulation was running, could you tell the **DelOperation** was delaying?

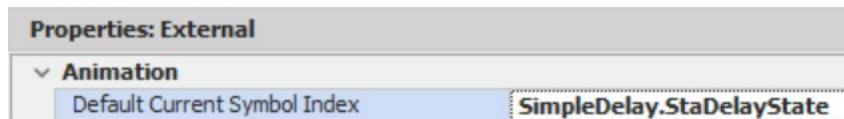
---

**Step 6:** Currently, only one symbol is defined. To add additional symbols to our simple delay, click first on the “External View” for the **SIMPLEDELAY** object and select the *Additional Symbols* tab. Now add an additional symbol and color the ellipse of the new one “green” (i.e., the green color will represent the status “Delaying,” meaning there are entities in the delay). See the symbols in Figure 20.11.



**Figure 20.11: Symbols for the Two States Idle and Delaying**

**Step 7:** To use the states to define which symbol to display, change the *Default Current Symbol Index* in the *Properties* window to the `SimpleDelay.StaDelayState`.



**Figure 20.12: Using the Delay State to Change the Symbol**

**Step 8:** Return to the primary **Model** and delete the current **DelOpeation** from your SIMIO model since the external view has not been updated with instances that have already been defined. Insert a new **SIMPLEDELAY** and set the delay time to “4” minutes. Save and run the model.

*Question 4:* Does it change color when entities occupy the delay?<sup>293</sup>

---

*Question 5:* What other embellishments to this simple delay object do you think would be helpful?

---

### Part 20.3: Embellishing with User-Defined Add-on Process Triggers

Perhaps the greatest flexibility you can add to your objects is to allow users to add on processes that change the object’s behavior to meet their needs. This addition assumes that your user knows how to create those processes. Many of the SIMIO standard objects allow the user to augment the processing of an object via add-on process triggers (e.g., *Run Initialized*, *Entered*, *Exited*, *Processing*, *Processed*, etc.). These user-defined add-on process triggers must be specified to allow our new object to mimic the standard objects.

**Step 1:** Select the **SIMPLEDELAY** object from the [Navigation] panel and navigate to the *Definitions*→*Properties* section. Properties associated with the add-on process triggers must be defined so the user can specify them. Four process triggers will be added (*Initialized*, *Entered*, *Exited*, *BeforeDelaying*, and *AfterDelaying*) using a “Process” property under the “Element Reference.”

---

<sup>293</sup> Note, the user has the ability change the *Current Symbol Index* in the model just like any other standard SIMIO object.

**Step 2:** Insert a PROCESS ELEMENT REFERENCE property for the “**Run Initialized**” with the following properties. Like the other objects, this process will run at the start of a simulation. New category names can be created by just typing in the category into the entry box.

Property	Value
Name	RunInitializedAddOnProcess
Description	Executes when the simulation run is initialized.
Required Value	False
Display Name	Run Initialized
Category Name	Add-on Process Triggers
Default Value	Empty String

- Add the “**Entered**” PROCESS ELEMENT REFERENCE property with the following properties that will be executed as soon as an object enters the **DELAYOBJECT**.

Property	Value
Name	EnteredAddOnProcess
Description	Occurs immediately after an entity has entered this object and before the delay.
Required Value	False
Display Name	Entered
Category Name	Add-on Process Triggers
Default Value	Empty String

- Add the “**Exited**” PROCESS ELEMENT REFERENCE property with the following properties that will be executed once an object leaves the **DELAYOBJECT**.

Property	Value
Name	ExitedAddOnProcess
Display Name	Exited
Description	Occurs immediately after an entity has exited this object.
Category Name	Add-on Process Triggers
Required Value	False
Default Value	Empty String

- Add the “**Before Delaying**” PROCESS ELEMENT REFERENCE property that will run immediately before the ENTITY starts the delay process, similar to the SERVER’ S *Before Processing* trigger.

Property	Value
Name	BeforeDelayingAddOnProcess
Display Name	Before Delaying
Description	Occurs immediately before an entity is to be delayed.
Category Name	Add-on Process Triggers
Required Value	False
Default Value	Empty String

- Finally, insert the “**After Delaying**” PROCESS ELEMENT REFERENCE property that will run once the ENTITY has finished delaying, similar to the SERVER’ S *After Processing* trigger.

Property	Value
Name	AfterDelayingAddOnProcess
Display Name	After Delaying

Description	Occurs immediately after an entity has been delayed
Category Name	Add-on Process Triggers
Required Value	False
Default Value	Empty String

**Step 3:** Return to the *Processes* tab and insert an *Execute* step into the **OnEnteredDelay** process, which allows processes to be executed, as seen in Figure 20.13. Specify the process to be the referenced property (*EnteredAddOnProcess*). The *Token Wait Action* property should be “WaitUntilProcessCompleted” under the *Advanced Options*, which forces the token to wait until the add-on process has completed running before it continues to the next step.

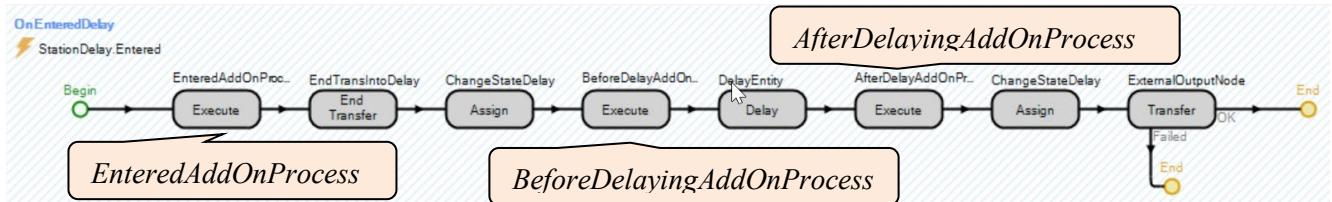


Figure 20.13: Specifying the Entered and Delayed Add-on Process Trigger

**Step 4:** Before the *Delay* step (see Figure 20.13) in the **OnEnteredDelay** process, insert another *Execute* step to run the process associated with the **BeforeDelayingAddOnProcess** trigger with the same (default) *Action* property value.

**Step 5:** After the *Delay* step, insert another *Execute* step that will run the process associated with the **AfterDelayingAddOnProcess** trigger with the same (default) *Action* property value.

**Step 6:** Add the **OnRunInitialized** Process, which is automatically defined for all **FIXED CLASS** objects by selecting it in the “Select Process” dropdown box. Insert an *Execute* step into the process as before, but specify the *RunInitializedAddOnProcess* referenced property as seen in Figure 20.14 and Figure 20.15.

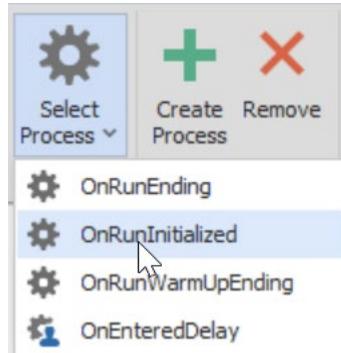
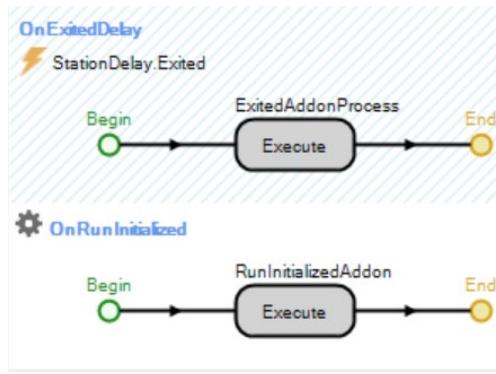


Figure 20.14: Using a Built-in Process for all Fixed Objects

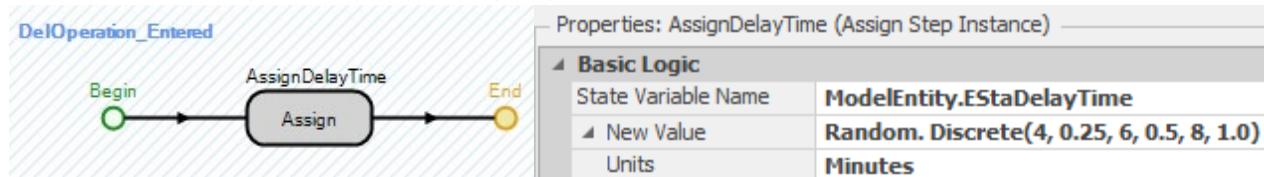
**Step 7:** Create a new Process named **OnExited** and specify the *TriggeringEvent* property as the *StationDelay.Exited* event. Again, insert an *Execute* step to execute the process for the user-defined *ExitedAddOnProcess* property.



**Figure 20.15: Adding an OnRunInitialized and OnExited Process**

**Step 8:** Let's utilize our new add-on process triggers as a “user” of our object to allow the delay time to vary for each entity (i.e., 25% of the time, it will be four minutes, 25% of the time it will be six minutes, and 50% of the time it will be eight minutes).

- Insert a new MODELENTITY real-valued state variable named EStaDelayTime whose Unit Type is Time, which will store the delay time.
- Specify the *Delay Time* of the **DelOperation** to be ModelEntity.EStaDelayTime.
- Navigate back to the **Model** and create a new *Entered Add-on Process* trigger for the **DelOperation**, as seen in Figure 20.16.
- Add an *Assign* step that assigns **ModelEntity.EStaDelayTime** a value from a Random. Discrete(4, 0.25, 6, 0.5, 8, 1.0) minutes.



**Figure 20.16: Assigning a Delay Time for Each Entity**

**Step 9:** Save and run the model for 100 hours, designating all entities for the new object.

**Question 6:** Theoretically, the time in the station should be 6.5 minutes (why?). What did you get for the “Holding” time in the **DelOperation**?

---

**Question 7:** How important are add-on processes in a simple object?

---

## Part 20.4: Embellishing with State Assignments

The SIMIO standard objects have state assignments that can be performed without the use of processes. This addition is considered a “convenience” to the user, especially since assignments are easily implemented in processes. Nonetheless, similar state assignments can be added to the new **SIMPLEDELAY** to illustrate how to add this capability. Recall that an *Assign* step may consist of several assignments. As a result, this addition must accommodate repeating specifications.

**Step 1:** State assignments are specified by a “repeating group property” because the number of assignments can vary. Add a *Repeat Group* property named **AssignmentsOnEntering** in the **SIMPLEDELAY** object with the properties specified in Figure 20.17. Again, type in the new category name **State Assignments** so it will be similar to the other standard objects.

Properties: AssignmentsOnEntering (Repeating Group Property)	
<b>Value</b>	
<b>Appearance</b>	
Display Name	<b>AssignmentsOnEntering</b>
Category Name	<b>State Assignments</b>
Expanded	False
Parent Property Name	
<b>General</b>	
Name	<b>AssignmentsOnEntering</b>
Description	<b>Optiona state assignments when entity is entering the object</b>
Required Value	<b>False</b>
Visible	True

Figure 20.17: Adding a Repeating Group Property

The repeat group allows you to specify multiple rows (i.e., repeating) of a set of properties (i.e., the state variable and the value expression).

**Step 2:** To create the properties that are to be repeated (i.e., grouped), first select the **Assignments OnEntering** group property, and then insert a new *State Standard Property* named **AssignmentsOnEnteringStateVariableName** since a state variable is used in an assignment with the properties specified in Figure 20.18.

Properties: AssignmentsOnEnteringStateVariableName (State Property)	
<b>Value</b>	
Default Value	<b>null</b>
Default Indices	0 Rows
Switch Property Name	
Switch Condition	Equal
Switch Value	
Reference Type	AnyState
Data Format	Real
<b>Appearance</b>	
Display Name	<b>State Variable Name</b>
Category Name	<b>Basic Logic</b>
Expanded	False
Parent Property Name	
<b>General</b>	
Name	<b>AssignmentsOnEnteringStateVariableName</b>
Description	<b>Name of the state variable that will be assigned a new value</b>
Required Value	<b>False</b>
Visible	True

Figure 20.18: Specifying a State Variable Property

**Step 3:** Next, insert the *expression* property that will specify the new value to be assigned to the state variable. Again, select the **AssignmentsOnEntering** group property first, and then insert a new *Expression Standard Property* named **AssignmentsOnEnteringNewValue** with the properties specified in Figure 20.19. Note the specification of the *Unit Type* carefully so units of the value can be specified based on the state variable.

Properties: AssignmentsOnEnteringNewValue (Expression Property)	
<b>Value</b>	
Default Value	<b>0.0</b>
Switch Property Name	
Switch Condition	Equal
Switch Value	
Candidate References	False
Unit Type	Unspecified
<b>Appearance</b>	
Display Name	<b>New Value</b>
Category Name	<b>Basic Logic</b>
Expanded	False
Parent Property Name	
<b>General</b>	
Name	<b>AssignmentsOnEnteringNewValue</b>
Description	<b>The new value to assign</b>
Required Value	<b>False</b>
Visible	True

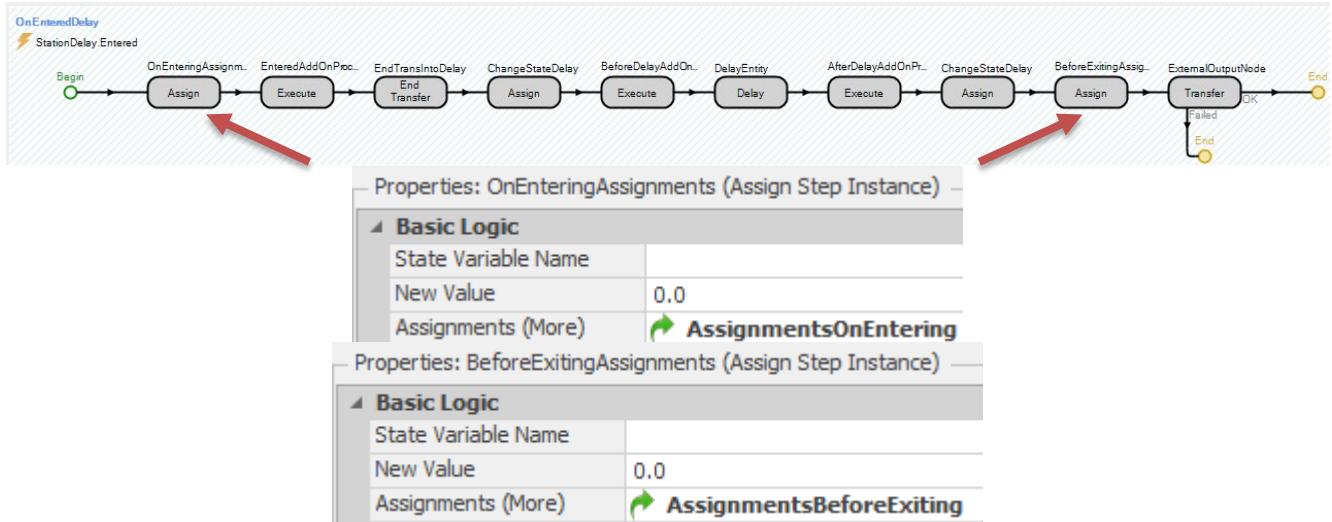
**Figure 20.19: Specifying the Expression Value Property of the Repeating Group**

**Step 4:** Repeat the process for another *Repeat Group Property* named **AssignmentsBeforeExiting**, as seen in Figure 20.20, which shows all the properties. Notice how the group properties have their own separate properties section. You can copy the **AssignmentsOnEntering** and just change the name.

Properties (Inherited)			
WorkDayExceptions.Properties (Inherited)			
WorkPeriodExceptions.Properties (Inherited)			
AssignmentsBeforeExiting.Properties			
<input checked="" type="checkbox"/> AssignmentsBeforeExitingStateVariableName	State Property	State Variable Name	Basic Logic
<input checked="" type="checkbox"/> AssignmentsBeforeEnteringNewValue	Expression Property	New Value	Basic Logic
AssignmentsOnEntering.Properties			
<input checked="" type="checkbox"/> AssignmentsOnEnteringStateVariableName	State Property	State Variable Name	Basic Logic
<input checked="" type="checkbox"/> AssignmentsOnEnteringNewValue	Expression Property	New Value	Basic Logic
Properties			
<input checked="" type="checkbox"/> DelayTime	Expression Property	DelayTime	Process Logic
<input checked="" type="checkbox"/> RunInitializedAddOnProcess	Process Element Property	Run Initialized	Add-on Process Triggers
<input checked="" type="checkbox"/> EnteredAddOnProcess	Process Element Property	Entered	Add-on Process Triggers
<input checked="" type="checkbox"/> ExitedAddOnProcess	Process Element Property	Exited	Add-on Process Triggers
<input checked="" type="checkbox"/> BeforeDelayingAddOnProcess	Process Element Property	Before Delaying	Add-on Process Triggers
<input checked="" type="checkbox"/> AfterDelayingAddOnProcess	Process Element Property	After Delaying	Add-on Process Triggers
<input checked="" type="checkbox"/> AssignmentsOnEntering	Repeating Group Property	AssignmentsOnEntering	State Assignments
<input checked="" type="checkbox"/> AssignmentsBeforeExiting	Repeating Group Property	AssignmentsBeforeExiting	State Assignments

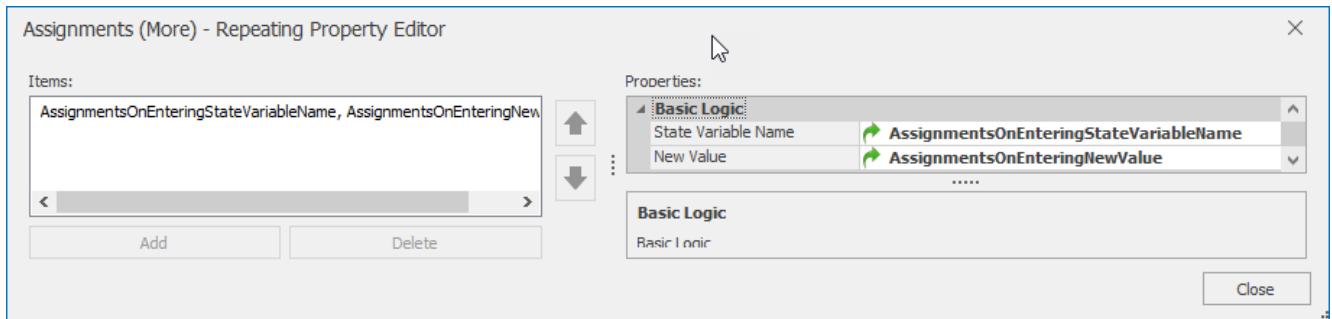
**Figure 20.20: Properties Associated with the SIMPLEDELAY**

**Step 5:** Once the *Repeating Group* properties have been defined, they can be used in *Assign* steps. Insert two *Assign* steps, as seen in Figure 20.21, before the “*OnEntering*” and “*BeforeExiting*” state assignments. One *Assign* step should be inserted before the execution of the entered add-on process and the other one should be inserted before the entity is transferred out of the node. Figure 20.21 shows how the reference properties are used to specify the *Before Exiting Assignments* property repeating group.



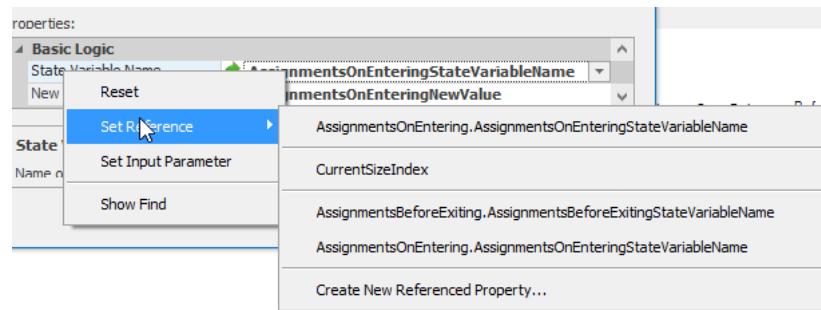
**Figure 20.21: Adding the Assign Process Steps for the Assignment States**

**Step 6:** For each of the *Assignments (More)* properties, the state variable and new value properties must be associated with the reference properties as well. Bring up the *Repeating Property Editor* by clicking the more button **AssignmentsOnEntering** ..., as seen in Figure 20.22.



**Figure 20.22: Setting up the State Variables and New Values to the Referenced Properties**

**Step 7:** For both the *State Variable Name* and *New Value* properties, set the reference property to the top property for each property. Figure 20.23 shows selecting the *State Variable Name* referenced property.

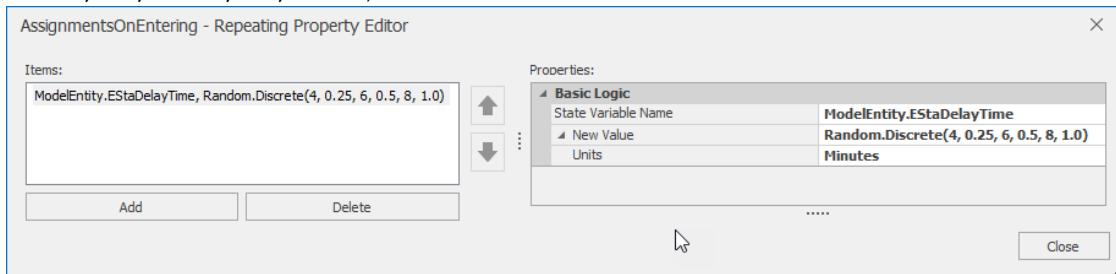


**Figure 20.23: Select the Top Reference Property for the State Variable Name**

**Step 8:** Let's utilize our new processes and assignments by specifying the delay time of each of the entities using the *OnEntering State Assignments*.

- First, remove the *Entered Add-on process* trigger by resetting it (reset or Null) so the add-on process will not specify the delay time.

- Repeat the same assignment as was done in Figure 20.24 using the **AssignmentsOnEntering State Assignments**. Assign the **ModelEntity.EStaDelayTime** a value from a Random.Discrete(4, 0.25, 6, 0.5, 8, 1.0) minutes.<sup>294</sup>



**Figure 20.24: Using the State Assignments Property**

**Step 9:** Save and run the model for 100 hours, designating all entities for the new object.

**Question 8:** What did you get for the Holding time in the **DelOperation**?

---

**Question 9:** How important are the state assignments in a simple object?

---

## Part 20.5: Adding Secondary Resources

While it is contrary to the purpose of a simple delay object, let's add the ability to have secondary resources, which could also be done via the add-on process triggers. We would require resources before the delay occurs and then release them right after the delay. Again, a *Repeat Group* property similar to the assignments must be utilized since the *Seize* and *Release* steps within the secondary resources can request/release more than one type of capacity.

**Step 1:** Recall the assignments had just two properties (i.e., the state variable and the value) associated with the *Repeat Group*. That is not the case for the *Seize* and *Release* steps, which have many properties. The easiest way is to sub-class the SERVER object and then copy and paste the secondary resources (e.g., *SecondaryResourcesSeizesBeforeProcessing* and *SecondaryResourcesReleasesAfterProcessing*) into the **SIMPLEDELAY**. Therefore, sub-class the SERVER by right-clicking to create a **MY SERVER** and then expand the Properties (Inherited) section.

**Step 2:** Select the **MY SERVER** from the [Navigation] window. Under the *Definitions*→*Properties* section, expand the **Properties(Inherited)**. Copy individually the three properties associated with **SecondaryResourceSeizesBeforeProcessing**, **SecondaryResourceSeizesBeforeProcessingMustSimultaneouslySeize**, and **SecondaryResourceSeizesBeforeProcessingImmediatelyTrySeize**, as seen in the red box in Figure 20.25. Then, paste this *Repeat Group* property and the two boolean properties into the **CAPACITYDELAY** properties.

---

<sup>294</sup> If the *Units* do not show, then the *New Value* property was not setup correctly (see Figure 20.19).

 SecondaryResourceSeizesBeforeProcessing	Repeating Group Property	Before Processing	S
 SecondaryResourceSeizesBeforeProcessingMustSimultaneouslySeize	Boolean Property	Must Simultaneously Seize	S
 SecondaryResourceSeizesBeforeProcessingImmediatelyTrySeize	Boolean Property	Immediately Try Seize	S
 SecondaryResourceSeizesAfterProcessing	Repeating Group Property	After Processing	S
 SecondaryResourceSeizesAfterProcessingMustSimultaneouslySeize	Boolean Property	Must Simultaneously Seize	S
 SecondaryResourceSeizesAfterProcessingImmediatelyTrySeize	Boolean Property	Immediately Try Seize	S
 SecondaryResourceReleasesOnEntering	Repeating Group Property	On Entering	S
 SecondaryResourceReleasesOnEnteringImmediatelyTryAllocateWhenReleased	Boolean Property	Immediately Try Allocate...	S
 SecondaryResourceReleasesBeforeProcessing	Repeating Group Property	Before Processing	S
 SecondaryResourceReleasesBeforeProcessingImmediatelyTryAllocateWhenReleased	Boolean Property	Immediately Try Allocate...	S
 SecondaryResourceReleasesAfterProcessing	Repeating Group Property	After Processing	S
 SecondaryResourceReleasesAfterProcessingImmediatelyTryAllocateWhenReleased	Boolean Property	Immediately Try Allocate...	S
 RunInitializedAddOnProcess	Process Element Property	Run Initialized	A

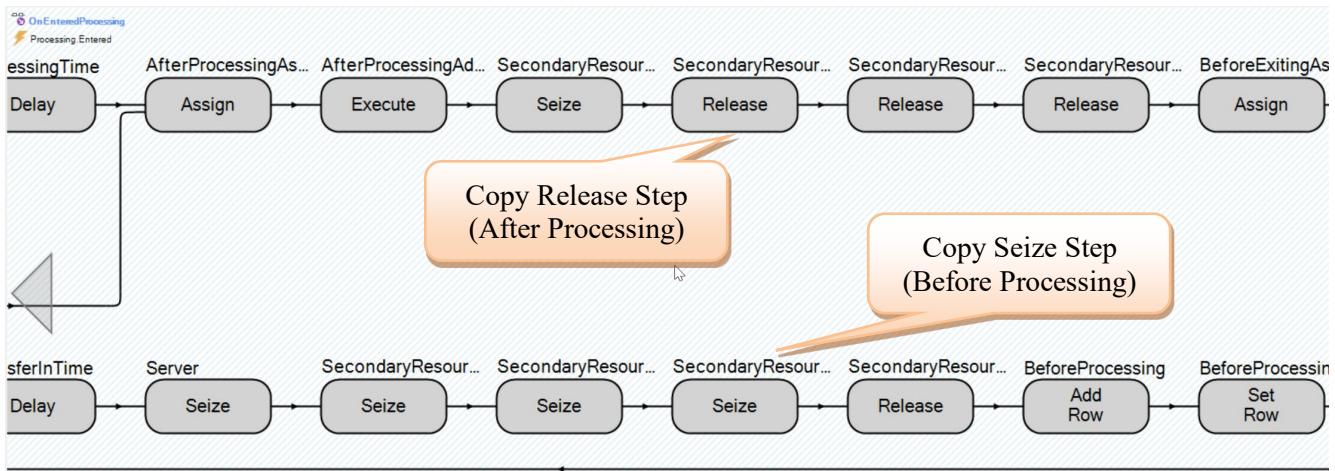
Figure 20.25: Copying the Before-Processing Seize and After-Processing Release Properties

**Step 3:** Repeat the process for the **SecondaryResourceReleasesAfterProcessing Repeat Group** property and the **SecondaryResourceReleasesAfterProcessingImmediatelyTryAllocateWhenReleased Boolean** property seen in the red box of Figure 20.25 and in the **SimpleDelay** properties of Figure 20.26.

> AssignmentsBeforeExiting.Properties			
> AssignmentsOnEntering.Properties			
<b>&lt; Properties</b>			
 DelayTime	Expression Property	DelayTime	Process Logic
 RunInitializedAddOnProcess	Process Element Property	Run Initialized	Add-on Process Triggers
 EnteredAddOnProcess	Process Element Property	Entered	Add-on Process Triggers
 ExitedAddOnProcess	Process Element Property	Exited	Add-on Process Triggers
 BeforeDelayingAddOnProcess	Process Element Property	Before Delaying	Add-on Process Triggers
 AfterDelayingAddOnProcess	Process Element Property	After Delaying	Add-on Process Triggers
 AssignmentsOnEntering	Repeating Group Property	AssignmentsOnEntering	State Assignments
 AssignmentsBeforeExiting	Repeating Group Property	AssignmentsBeforeExiting	State Assignments
 SecondaryResourceSeizesBeforeProcessing	Repeating Group Property	Before Processing	Secondary Resources/Other Resource Seizes
 SecondaryResourceSeizesBeforeProcessingMustSimultaneouslySeize	Boolean Property	Must Simultaneously Seize	Secondary Resources/Other Resource Seizes
 SecondaryResourceSeizesBeforeProcessingImmediatelyTrySeize	Boolean Property	Immediately Try Seize	Secondary Resources/Other Resource Seizes
 SecondaryResourceReleasesAfterProcessing	Repeating Group Property	After Processing	Secondary Resources/Other Resource Releases
 SecondaryResourceReleasesAfterProcessingImmediatelyTryAllocateWhenReleased	Boolean Property	Immediately Try Allocate When Released	Secondary Resources/Other Resource Releases
<b>&lt; SecondaryResourceReleasesAfterProcessing.Properties</b>			
 SecondaryResourceReleasesAfterProcessingResourceType	Enumeration Property	Resource Type	Resource Information
 SecondaryResourceReleasesAfterProcessingResourceName	Object Property	Resource Name	Resource Information
 SecondaryResourceReleasesAfterProcessingResourceListName	Object List Property	Resource List Name	Resource Information
 SecondaryResourceReleasesAfterProcessingQuantityType	Enumeration Property	Quantity Type	Required Quantity & Constraints
 SecondaryResourceReleasesAfterProcessingNumberofResources	Expression Property	Number Of Resources	Required Quantity & Constraints
 SecondaryResourceReleasesAfterProcessingUnitsPerResource	Expression Property	Units Per Resource	Required Quantity & Constraints
 SecondaryResourceReleasesAfterProcessingReleaseOrder	Enumeration Property	Release Order	Required Quantity & Constraints
 SecondaryResourceReleasesAfterProcessingSelectionCondition	Expression Property	Selection Condition	Required Quantity & Constraints
 SecondaryResourceReleasesAfterProcessingKeepReservedCondition	Expression Property	Keep Reserved If	Advanced Options
 SecondaryResourceReleasesAfterProcessingReservationTimeout	Expression Property	Reservation Timeout	Advanced Options
 SecondaryResourceReleasesAfterProcessingOnReleasedProcess	Process Element Property	On Released Process	Advanced Options
 SecondaryResourceReleasesAfterProcessingSkipReleaseCondition	Expression Property	Skip Release If	Advanced Options
<b>&lt; SecondaryResourceSeizesBeforeProcessing.Properties</b>			

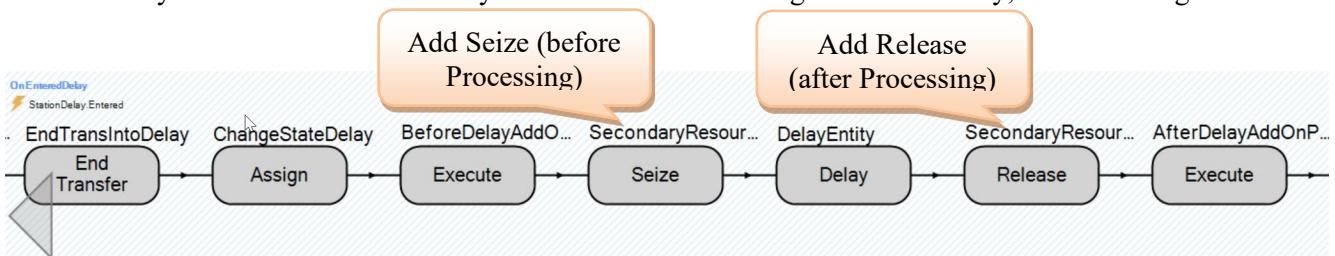
Figure 20.26: Repeat Group Properties for Allowing Secondary Resources

**Step 4:** From the **OnEnteredProcessing** process (see Figure 20.27) inside the **MySERVER** object, copy the **Seize** and **Release** steps before and after the process.



**Figure 20.27: Copy Seize and Release Steps from the MyServer's OnEnteredProcessing Process**

**Step 5:** Now, for into the **OnEnteredDelay** process of the **SIMPLEDELAY** object, add the first seize secondary resources before the delay and then release them right after the delay, as seen in Figure 20.28.



**Figure 20.28: Adding the Seize and Release into the OnEntered**

**Step 6:** Return to the **Model** and insert a RESOURCE named **ResService** with a capacity of two, which is required before the delay can occur.

**Step 7:** Change the arrival rate to 3.5 minutes and run the model for 100 hours.

**Question 10:** For a simulation of 100 hours, what is the holding time in the **StationDelay**?

**Step 8:** In the **Secondary Resources** of the **DelOperation**, seize the **ResService** in the *Other Resource Seizes→Before Processing* and release it in the *Other Resource Releases→After Processing*.

**Step 9:** Save and run the model interactively, observing the animation.

**Question 11:** Does the animation behave as you expect?

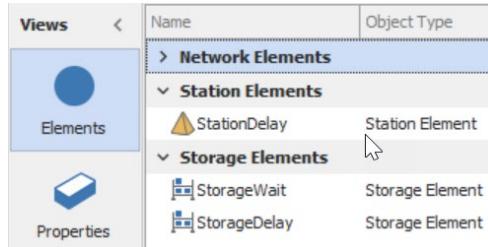
**Question 12:** For a simulation of 100 hours, what is the holding time in the **StationDelay**?

## Part 20.6: Using Storages to Distinguish Waiting versus Delaying

The processing queue for the delay station now includes waiting for secondary resources and the actual delay, so its holding time increases. Since the delay station currently has an infinite capacity, all the entities are shown in the processing animated queue even though they are not delayed. The **SERVER** object utilizes two

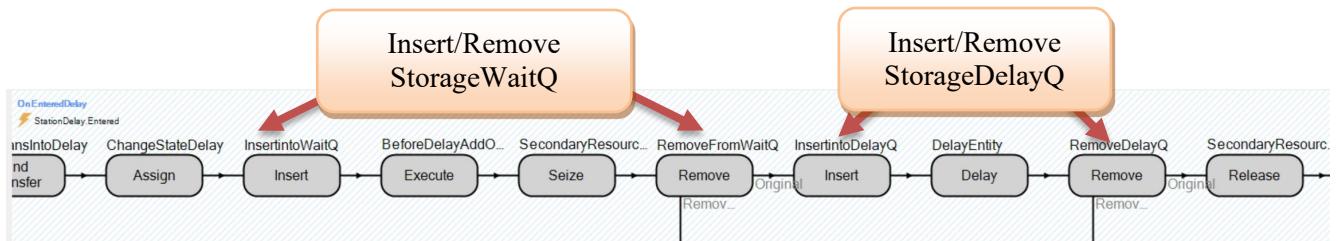
stations (i.e., **InputBuffer** and **Processing**) to decouple the waiting and processing times. Another approach is to utilize **STORAGE** elements, which automatically calculate statistics used in the DMV problem in Chapter 8.

**Step 1:** Select the **SimpleDelay** from the [Navigation] panel. From the *Definitions*→*Elements* section, insert two **STORAGE** elements named **StorageWait** and **StorageDelay**, as in Figure 20.29.



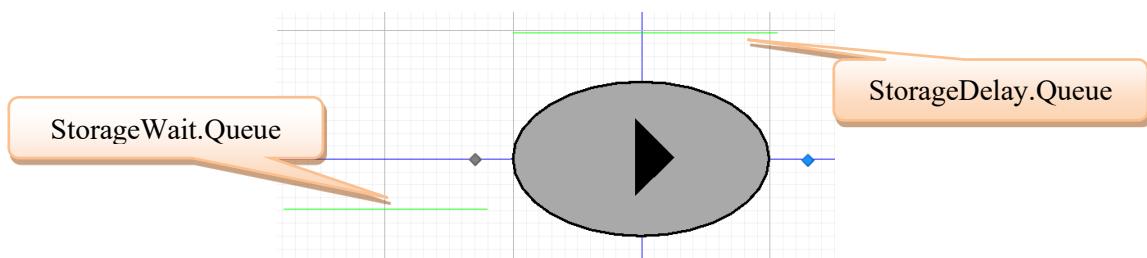
**Figure 20.29:** Adding Two `STORAGE` Elements to Keep Track of Waiting and Delaying

**Step 2:** To calculate how long entities waited for secondary resources, we need to insert them into the **StorageWait** before we seize the resources and then remove them after the resources have been seized. We repeat the process to keep track of the actual delay time by inserting the entities into the **StorageDelay** right before the delay and removing them right after the delay, as seen in Figure 20.30.



**Figure 20.30:** Utilizing the StorageWait and StorageDelay to Track Waiting Times

**Step 3:** Finally, modify the **SIMPLEDELAY** object “External” view. Insert a new animated queue that references **StorageWait.Queue** while the processing queue should reference the **StorageDelay.Queue**.



**Figure 20.31: Adding the Delay and Wait Queues to the External View**

**Step 4:** Save and run the **Main** model for 100 hours.

*Question 13:* When you switch back to the **Main** model, are the two new queues visible?

*Question 14:* On average, how long did parts wait for the **ResService**?

*Question 15:* How long were parts actually delayed in the DelOperation?

*Question 16:* Comment on the simplicity and transparency of this object.

**Step 5:** At this point, we have replicated a lot of the SERVER object, which is a complex object (i.e., a heavyweight/bloated object) versus our SIMPLEDELAY (i.e., lightweight) object. Therefore, we will check our current delay with that of the SERVER, as seen in Figure 20.32.

- Delete **TDelay** (i.e., **DELAYTRANSFERNODE**) and insert a SERVER named **SrvComparison**, which is connected via connectors from the **Output@SrcParts** and **ParentOutput@SepCopy** and to the **SnkExit**. Set the *Processing Time* property to `ModelEntity.EStaDelayTime` and Initial Capacity to “Infinity.”
- Delete the current **DelOperation** and insert another SIMPLEDELAY to get the new external view (i.e., queues). Set the *Delay Time* property to `ModelEntity.EStaDelayTime`. Use connectors to connect it to the two nodes as well as directly from the **Output@SrcParts**.
- Remove the **SetTime** add-on process trigger from the **TNormal**.
- Change the two TIMEPATHS that leave the **MemberOutput@SepCopy** and the *one connecting to the SnkExit* to CONNECTORS.

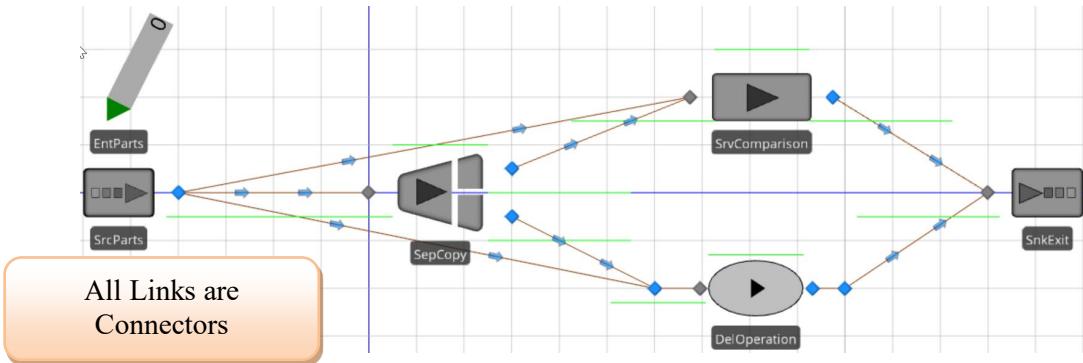


Figure 20.32: Model to Compare SimpleDelay and the SrvComparison

**Step 6:** For the **DelOperation\_Entered** process, change the *Assign* step to set the entity’s delay time to `Random.Pert(1, 2, 4, 1)`.<sup>295</sup>



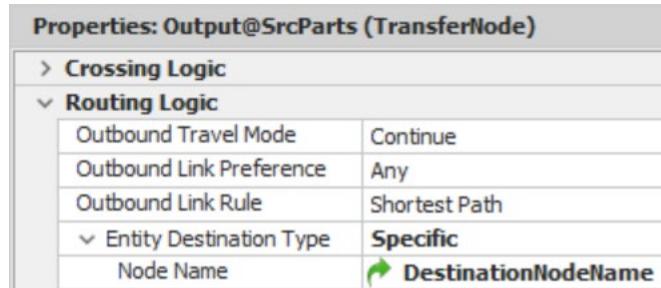
Figure 20.33: Changing the Delay time of the ModelEntity

**Step 7:** Next, for the **SrcParts**, specify the **DelOperation\_Entered** as the *Created Entity* add-on process trigger and change the arrival *Interarrival Time* to `Random.Exponential(3, 2)`.

**Step 8:** To facilitate comparing the two systems (i.e., the server versus the simple delay) in an experiment, we will route the entities either to the **SrvComparison** or to the **DelOperation**. Select the **Output@SrcParts**, change the *Entity Destination Type* to “Specific,” and then right-click on the *Node Name* to create a new reference property, as seen in Figure 20.34.

---

<sup>295</sup> Note we are using the fourth parameter which specifies the random stream so we induce as much correlation into comparing scenarios.



**Figure 20.34:** Setting up the Output Node for Experimentation

**Step 9:** Create a SIMIO experiment where we will specify the Destination Node as the only control.

- Set the run length to 5000 hours. Also, specify that you only want “1” replication.
- Specify the **Input@DelOperation** as the *DestinationNodeName* and run the one replication observing the actual run time.
- Reset the experiment, change the *DestinationNodeName* to **Input@SrvComparison**, and rerun it.

*Question 17:* What did you get for the execution time for each of the experiments? Do you think the difference is important?

---

*Question 18:* What factors influence this execution time comparison?

---

*Question 19:* Although execution time is better for our simple delay object versus a server object, is execution time the most important consideration when creating a simulation object?

---

**Step 10:** Now let us compare the two systems, but first, we need to add secondary resources to both objects, which we will do via add-on process triggers to force some waiting to occur.

- Create a new process named **SeizeService** that utilizes a *Seize* step to acquire the **ResService**.
- Create a new process named **ReleaseService** that will release the **ResService**.
- Specify the **SeizeService** as the add-on process trigger for *Before Processing* for the **SrvComparison** and the *Before Delaying* for the **DelOperation**.
- Finally, specify the **ReleaseService** as the add-on process trigger for *After Processing* for the **SrvComparison** and the *After Delaying* for the **DelOperation**.

**Step 11:** From the Experiment window, set up two scenarios that specify **Input@DelOperation** and **Input@SrvComparison**, respectively, with 100 replications. Change the run length back to 100 hours.

**Step 12:** Save and run the experiment.

*Question 20:* What is the average and half-width of the total time in the system and the time waiting for the two scenarios?

---

## Part 20.7: Considering Capacity Like a Server

At this point, we have added the ability to have secondary resources to constrain the capacity of the delay. Adding capacity to our delay object adds new considerations because entities cannot enter a delay when the delay object is at capacity.

**Step 1:** Select the **SIMPLEDELAY** object and access the properties. Rename our **SIMPLEDELAY** object.<sup>296</sup> to **CAPACITYDELAY** and change the *Resource Object* property in the “Advanced Options” to “True.” We will use the capacity specifications that are a consequence of making this a resource object (note the “additions” to the “Process Logic” category).

**Step 2:** The **FIXED OBJECT** now provides the ability to change the capacity. However, the **StationDelay** station is the location that actually constrains the entities. Therefore, from the *Definitions→Elements*, specify the properties in Figure 20.35 to link the **CapacityDelay** properties, which are visible to the modeler, to the internal **STATION**’s properties.

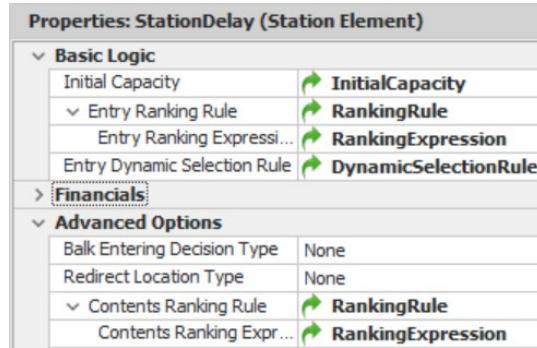


Figure 20.35: Linking the Station Capacity to the CapacityDelay

**Step 3:** The **OnEnteredDelay** process can now be modified, as shown in Figure 20.36.

- Insert a *Seize* step after the secondary resources seize. To seize capacity from the object, specify **PARENTOBJECT** as the *Object Type* within the *Seize* step, as seen in Figure 20.37.
- Since the object is constraining the entities, we will move the *Assign* that changes the state to delaying after the *Seize* of the **CAPACITYDELAY** object.

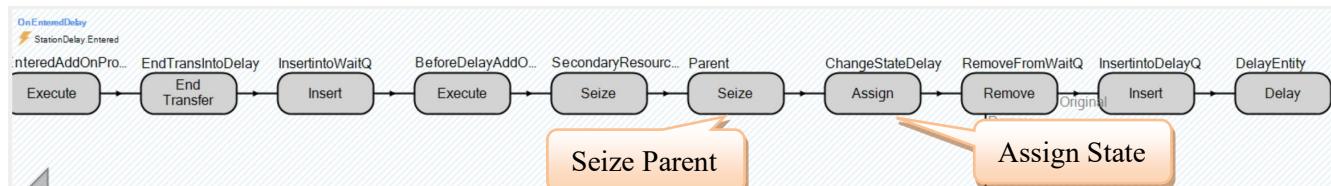


Figure 20.36: OnEnteredDelay CAPACITYDELAY Process

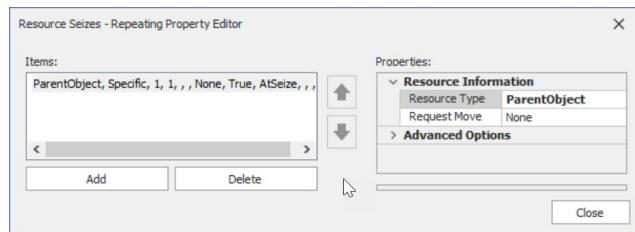
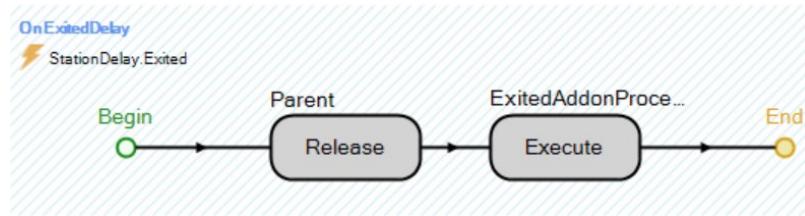


Figure 20.37: Seizing the Parent Object

**Step 4:** To release the object, insert a *Release* step into the **OnExitedDelay** using the same object type as in the *Seize* step.

<sup>296</sup> You could have sub-classed the **SIMPLEDELAY** to be the **CAPACITYDELAY**.



**Figure 20.38: Releasing the CAPACITYDELAY Object in OnExitedDelay**

**Step 5:** In the **Model**, set the *Initial Capacity* property to “2” for both the **SrvComparison** and the **DelOperation**. Also, remove all the add-on process triggers for both objects.

**Step 6:** Save and run the experiment.

*Question 21:* What is the average and half-width of total time in the system and the time waiting for the two scenarios (compared to the previous section)?

---

*Question 22:* What is the average time (in minutes) spent waiting in the **DelOperation** Entry Queue?

---

*Question 23:* Why do you think the entities are stuck waiting in the Entry queue for the **DelOperation**?

---

*Question 24:* What is the average holding time spent processing or delaying (in minutes)?

---

**Step 7:** You will also notice that the utilization of each object’s resource is the same, and the state utilization of the **SERVER** versus the **CAPACITYDELAY** (i.e., **Starving** versus **Idle** and **Processing** versus **Delaying**). To fix the issue of not seeing the entities waiting, we can change the animated queue that is set to the **StorageWait.Queue** or add an additional animated queue that sets as the *Queue State* property to **DelayStation.EntryQueue**. Note, if we added an additional queue, we could distinguish among waiting for the actual object (i.e., **ENTRY** queue statistics) versus waiting for secondary resources (i.e., **STORAGEWAIT** queue statistics).

*Question 25:* What are some benefits, other than speed, relative to this new **CapacityDelay** object?  
Consider modeling convenience, learning, and transparency.

---

## Part 20.8: Commentary

- Creating your own specialized objects can really enhance your models, which is one of the important advantages of using SIMIO. However, if you are only going to use the object once in one particular model, it may not make sense to spend time creating a specialized object that could be handled in processes. The advantage is that it can be reused within the same model or different models.
- Instead of the need for capacity at the station (location), consider the capacity restriction of the object to be a separate concern from the resources needed. It greatly simplified the object, especially in contrast to the SIMIO server.
- However, as we developed the object, we realized there was a strong interest in adding “features,” and the appropriate stopping point was unclear.

# Chapter 21

## Continuous Variables, Reneging, Interrupt, Debugging: A Gas Station

---

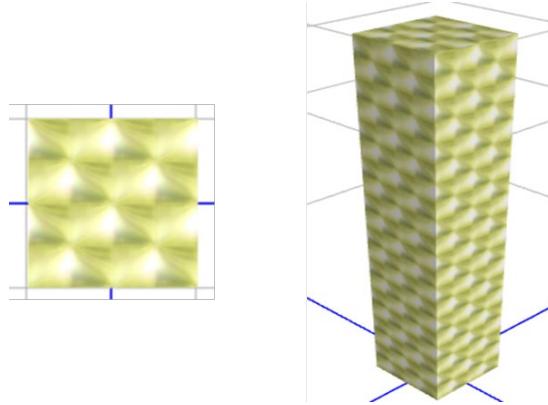
Continuous variables are variables whose value can change continuously during the simulation (as opposed to discrete variables). Continuous variables in SIMIO can be “Level” or “Level with Acceleration” state variables – see “States” under the *Definitions* tab in the “Continuous” section. When a `LEVEL` state variable is created, you may also specify the rate of change, called the *Initial Rate Value* property, and its initial state, which is called the *Initial State Value* property. `LEVEL WITH ACCELERATION` variables in addition to specifying the rate of change and the initial state, you may also specify the acceleration rate, called the *Initial Acceleration Value* property, as well as the length of the acceleration via the *Initial Acceleration Duration Value*. Unlike discrete state variables, continuous state variables can change continuously based on the rate and acceleration.

### Part 21.1: Simple Tank<sup>297</sup> Manual Process

**Step 1:** Create a new model. We are going to manage a simple gasoline tank by changing its rate value. We will assume the tank volume is between 0 and 100 gallons.

**Step 2:** Add a fixed `RESOURCE` object to the model named **ResTank**, which will represent the tank.

**Step 3:** Create a new symbol named **TankSymbol** to associate with the tank (from the *Project Home*→*New Symbol*→*Create Symbol*), such as the one in Figure 21.1. The symbol that represents the interior is a square with a height of ten meters (whose height determines the content/level of the tank). Add a *Gold Comb* texture to give it the appearance of gasoline for both the top and the sides.



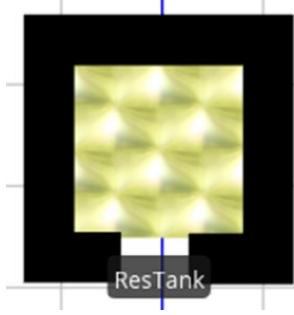
**Figure 21.1: 2D and 3D Representation of the Tank Symbol**

**Step 4:** Next, use the **TankSymbol** as the symbol for the **ResTank** `RESOURCE`.<sup>298</sup> Once this is done, draw a static “case” around it so you can see the level of the tank in 3D, as shown in Figure 21.2. The case is made from a thick “Polyline,” leaving a small opening in the front to see the tank’s level. The polyline also needs a height of ten meters.

---

<sup>297</sup> This “simple tank” example is adapted from the SIMIO “SimBits” which are another way help you learn about SIMIO.

<sup>298</sup> Select the `RESOURCE` object and then choose the symbols from the “Symbols” tab.



**Figure 21.2: Tank Symbol**

**Step 5:** Now add a LEVEL continuous state variable named **GStaLevelConTank** from the *Definitions→States* section. Let's set the *Initial State Value* to 0 and the *Initial Rate Value* to 0. A negative rate value means the tank is being drained, while a positive value means it is being filled.

**Step 6:** When we need to modify a parameter for a model, it is convenient to specify a model property rather than specifying a constant. Doing so also helps in eliminating mistakes by using the same symbolic name for a particular value. Therefore, a minimum and maximum level for the tank should be defined as a standard real property named **TankMin** and **TankMax**.<sup>299</sup> For these two properties, the *Unit Type* should be “Volume” and *Default Units* equal to “Gallons.” Add two “Expression” standard properties named **RatePositive** and **RateNegative** to represent the rate at which the tank will fill up and the rate at which the rate will be emptied. For these two properties, the *Unit Type* should be “VolumeFlowRate” and *Default Units* equal to “Gallons per Hour.” Place each property into a new **Tank Category**.

**Step 7:** Next, right-click on the **Model** in the [Navigation] panel to access the model properties and set the values of the minimum level property (**TankMin**) to 0, the maximum level property (**TankMax**) to 100, the fill rate (**RatePositive**) to 100, and the emptying rate (**RateNegative**) to -200.

**Step 8:** Now add two MONITOR Elements named **MonTankEmpty** and **MonTankFull**. These MONITORS will detect the crossing of the **GStaLevelConTank** at the **TankMin** and **TankMax** levels, respectively.

**Step 9:** For the **MonTankEmpty** MONITOR, set the properties as shown in Figure 21.3.

Properties: MonTankEmpty (Monitor Element)	
Basic Logic	
State Variable Name	<b>GStaLevelConTank</b>
Monitor Type	<b>CrossingStateChange</b>
Crossing Direction	<b>Negative</b>
Initial Threshold Value	<b>TankMin</b>
Monitored State Variables (More)	0 Rows
Trigger Condition	
Triggered Process Name	<b>MonTankEmpty_OnChangeDetectedProcess</b>

**Figure 21.3: TankEmpty Monitor Specification**

**Step 10:** And for the **MonTankFull** MONITOR, see Figure 21.4 for the property settings.

---

<sup>299</sup> This is done under the *Definitions* tab in the *Properties* section..

Properties: MonTankFull (Monitor Element)	
Basic Logic	
State Variable Name	GStaLevelConTank
Monitor Type	CrossingStateChange
Crossing Direction	Positive
Initial Threshold Value	TankMax
Monitored State Variables (More)	0 Rows
Trigger Condition	
Triggered Process Name	MonTankFull_OnChangeDetectedProcess

Figure 21.4: MonTankFull Monitor Specification

**Step 11:** Create new processes for the two *On Change Detected Process* add-on process triggers, which are defined in Figure 21.5 and Figure 21.6.<sup>300</sup>

**Step 12:** The *MonTankEmpty\_OnChangeDetectedProcess* process is called when the tank empties (i.e., the GStaLevelConTank reaches zero). We will need to stop emptying the tank, have a small delay before beginning to fill, and then start filling, as seen in Figure 21.5.



Figure 21.5: When the Tank Empties

- The first *Assign* step sets both the **GStaLevelConTank.Rate** to “0,” which stops the **LEVEL** state variable from changing and sets the **ResTank.Size.Height** to “0”.
- Use the *Delay* step to force a 0.2 hour delay before the tank begins to fill again.
- The final *Assign* step sets the **GStaLevelConTank.Rate** to its maximum value of **RatePositive** which will allow the tank to start to fill since it is a positive value.

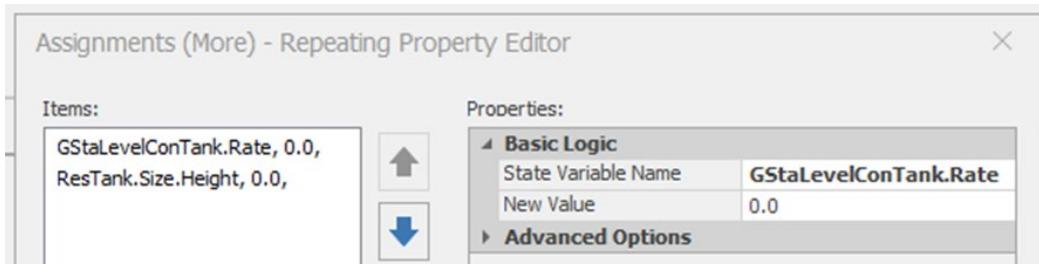


Figure 21.6: Assigning the Rate and the Height

**Step 13:** The *MonTankFull\_OnEventProcess* is called when the tank fills to the maximum level. Once it reaches this level, stop filling, force a small delay before allowing it to empty, and then allow the emptying to start, as seen in Figure 21.7.



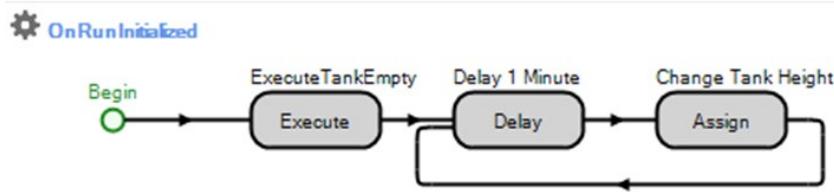
<sup>300</sup> You can either create the process here that will respond to the MONITOR event or you can create a generic process with the *Triggering Event* property set to the MONITOR event (e.g., MonTankFull.Event).

**Figure 21.7: When the Tank Fills**

- The *Assign* step sets the **GStaLevelConTank.Rate** to 0. A 0.1 hour delay is assumed before the tank can start emptying, and then the last *Assign* step will set the rate to a negative value to allow emptying (i.e., set the **GStaLevelConTank.Rate** to the **RateNegative**).

**Step 14:** Finally, we need to take care of the “height” of the tank animation. We will do that in the *OnRunInitialized* process for the model, which will run when the simulation starts. From the “Processes” tab, select the *OnRunInitialized* process from the “Select Process” drop-down.

- This process should *Execute*<sup>301</sup> the **MonTankEmpty\_OnEventProcess** once to initialize the tank for filling. Then, it will *Delay* for one minute and then *Assign* the **ResTank.Size.Height** to **GStaLevelConTank / (TankMax/10)** as in Figure 21.8.



**Figure 21.8: Changing the Height of the Tank**

- Notice that the system loops continuously every minute, updating the tank height for animation purposes. Do this by grabbing the “*End*” point and dropping it on top of the *Delay* step. Therefore, the TOKEN will continually loop through these steps until the simulation stops.

**Step 15:** For the tank level to show in the animation, the *Current Symbol Index* in the “Animation” section of the **ResTank** resource must be set to zero, so the *OnRunInitialized* process is the only specification to change the height. Select the **ResTank RESOURCE**, add an additional symbol, and then set the *Current Symbol Index* property to zero.

**Step 16:** To complete our animation, add six STATUS LABELS from the *Animation* tab, as seen in Figure 21.9, where the expressions are **GStaLevelConTank.Rate\*264.1720524**, **GStaLevelConTank \*264.1720524**, and **ResTank.Size.Height** respectively<sup>302</sup>. Also, for the word Status labels, change the background to white.

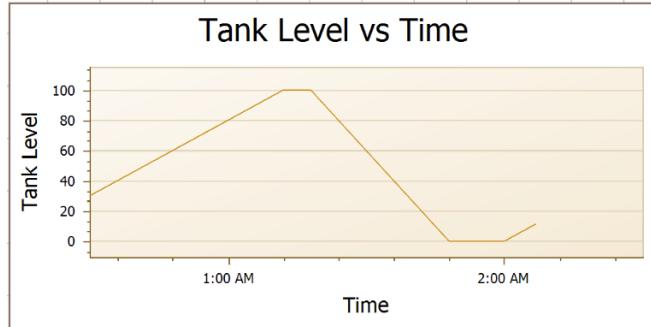


**Figure 21.9: Adding Status Labels**

**Step 17:** Next, add a Status Plot of **GStaLevelConTank** vs. Time (hours) – see Figure 21.31. Change the Title, *x-axis* label, *y-axis* label, *text scale*, and the time range to plot (i.e., 2 hours) from the *Appearance* tab.

<sup>301</sup> The *Execute* step will run any process that has been defined.

<sup>302</sup> The default unit for volume and flow rate is cubic meters and cubic meters per hour which we are converting back to gallons and gallons per hour.



**Figure 21.10: Status Plot**

**Step 18:** Also, a Circular Gauge (from the “Animation” panel) may be placed on top of the tank, as seen in Figure 21.11.<sup>303</sup> It is a way to visualize the tank level in 2D. After adding the circular gauge with the correct style, under the *Scale* section’ set the *Minimum* property to zero, the *Maximum* property to 100, and the *Ticks* to 11. The *Expression* should be the level state of the **GStaLevelConTank**.



**Figure 21.11: Adding a Circular Gauge the Current Tank Level**

**Step 19:** Save and run the model, observing the status labels and plots. Also, switch to 3D to see the tank content rise and fall.

*Question 26:* Do the animations and plots appear appropriate?

---

*Question 27:* Why does the tank empty faster than filling up?

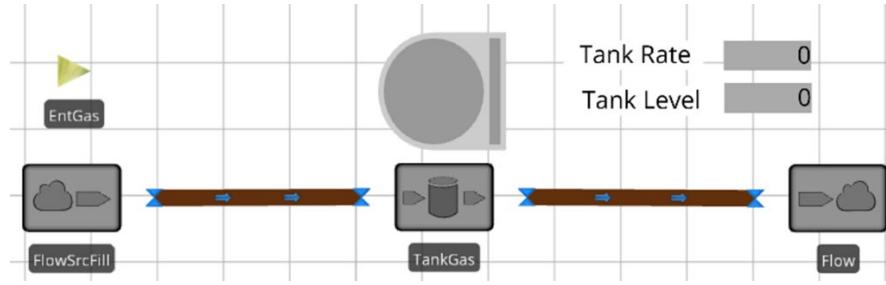
---

## Part 21.2: Simple Tank Revisited using the Flow Library

SIMIO provides a [Flow Library] to assist in handling pipes (i.e., flows) and tanks. The library provides a FLOW SOURCE for creating flow entities, FLOW SINK, TANK, FLOWNODE (i.e., regulator), and FLOW CONNECTOR used to connect FLOWNODES. We will model the simple tank again using the [Flow Library], as seen in Figure 21.12

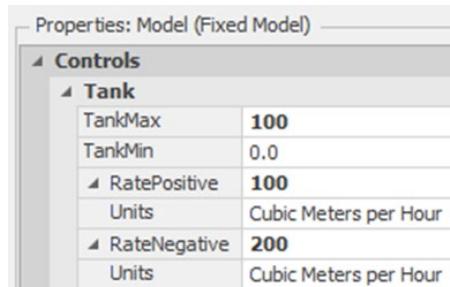
---

<sup>303</sup> Move to the 3D view and use the shift key to move the gauge up in space to place it directly on top.



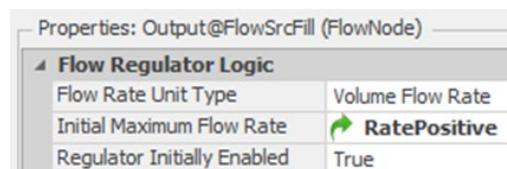
**Figure 21.12: Simple Tank using the SIMIO Flow Library**

**Step 1:** Insert a new fixed model named **TankFlow**.<sup>304</sup> Copy the four properties (i.e., **TankMax**, **TankMin**, **PositiveRate**, and **NegativeRate**) from the previous model and set their values as seen in Figure 21.13.<sup>305</sup>



**Figure 21.13: Setting up the Prosperities**

**Step 2:** Insert a new FLOWSOURCE named **FlowSrcFill**, which will be used to fill the tank.<sup>306</sup> Select the output FLOWNODE **Output@FlowSrcFill** and change the *Initial Maximum Flow* to **RatePositive** (see Figure 21.14).



**Figure 21.14: Setting the Correct Unit of the Output Flow**

**Step 3:** Insert a new MODELENTITY named **EntGas**. Change the texture of the entity to the *Gold Comb* texture as before to cause the tank to fill up with gas.

**Step 4:** Next, add a TANK named **TankGas**, which will hold the gasoline flowing from the FLOWSOURCE. Set the *Initial Volume Capacity* and *High Mark* to **TankMax** as well as the *Low Mark* to **TankMin**. Tanks have the ability to specify the capacity as well as have five different level marks (i.e., low-low, low, middle, high, and high-high marks) which will trigger processes when the tank reaches these levels. You should shrink the size of the tank animation to be smaller so you can see the tank filling and emptying easier.

<sup>304</sup> You can also just delete the two MONITOR elements, the three processes, the circular gauge, the ResTank and polyline from the current model rather than copying the objects over to the new model.

<sup>305</sup> Note, the rate negative is positive this time because of the way the flow library works.

<sup>306</sup> The regulator inside the **Output@FlowSrcFill** that embodies the continuous level state variables and thus the rates as was done before.

Properties: TankGas (Tank)	
Flow Storage Logic	
Capacity Unit Type	Volume
Initial Volume Capacity	TankMax
Initial Contents	0 Rows
Auto Refill Mode	No Automatic Refills
Purge Contents Triggers	0 Rows
Clean-In-Place Triggers	0 Rows
Tank Level Marks	
Low-Low Mark	0.0
Low Mark	TankMin
Mid Mark	0.0
High Mark	TankMax
High-High Mark	0.0

Figure 21.15: Setting up the 100 Gallon Tank

**Step 5:** Select the **Output@TankGas** FLOWNODE and set the *Initial Maximum Flow Rate* to 200 gallons per hour, which represents the emptying rate.<sup>307</sup> Also, set the *Regulator Initially Enabled* to “False” to indicate the regulator is closed and, therefore, not flowing (i.e., emptying), as seen in Figure 21.16.

Properties: Output@TankGas (FlowNode)	
Flow Regulator Logic	
Flow Rate Unit Type	Volume Flow Rate
Initial Maximum Flow Rate	RateNegative
Initial Output Yield Factor	1.0
Initial Output Entity Type	
Regulator Initially Enabled	<b>False</b>

Figure 21.16: Specifying the Tank Emptying Rate

**Step 6:** Add a FLOWSINK named **FlowSink** to destroy the flow of gas emptying the tank.

**Step 7:** Connect the **FlowSrcFill**, **TankGas**, and **FlowSink** using FLOWCONNECTORS.

**Step 8:** Copy the Tank Rate and Tank Level status labels as well as the status plot from the previous model. Change the expressions to `TankGas.FlowContainer.Contents.Volume.Rate * 264.1720524` and `TankGas.FlowContainer.Contents.Volume * 264.1720524` for the rate and tank level, respectively. Let the plot expression be the same as the tank-level expression.

**Step 9:** Save and run the model. You may need to modify the speed factor to 100 to see the tank level rising.

**Question 28:** What happens when the simulation runs with regard to the tank and the two FLOWCONNECTORS?

---

**Question 29:** What did you notice about the tank rate when the tank reached full capacity?

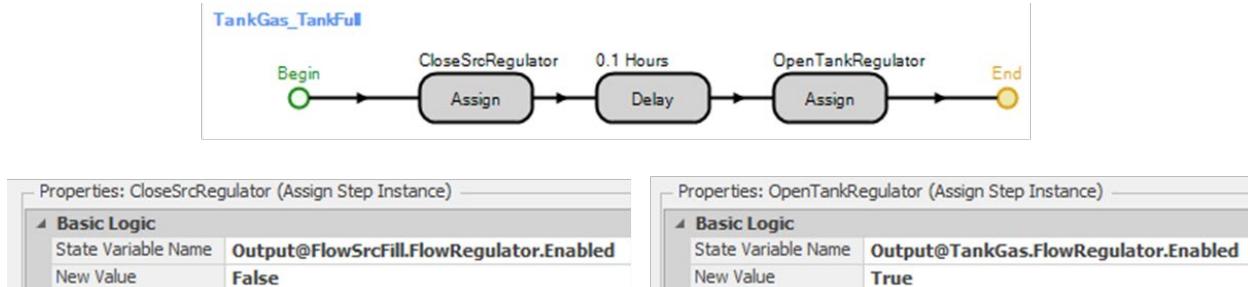
---

**Question 30:** Does the tank empty once it fills?

---

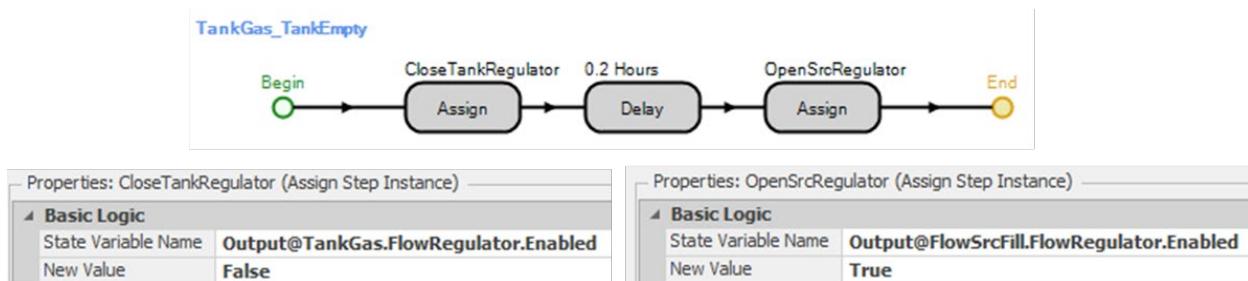
<sup>307</sup> Notice we do not specify it to be negative as was done in the manual process of the previous section.

**Step 10:** Once the tank fills up, the flow from the **FlowSrcFill** stops flowing, but the tank does not empty as the **Output@TankGas** regulator is not enabled. When the tank fills up, we need to delay for 0.1 hours and then open up this regulator while shutting down the regulator at **Output@FlowSrcFill**. Under the *Tank Level Rising* category, insert the *Tank Full* Add-on Process Trigger, as seen in Figure 21.17.<sup>308</sup> The FLOWNODES contain a REGULATOR element, which can be opened and closed to control the flow.



**Figure 21.17: Shutdown the Flow Source and then Open up the Tank Output**

**Step 11:** Once the tank empties, we need to shut down the outflow regulator of the tank, delay for 0.2 hours, and then open up the regulator at **Output@FlowSrcFill** to start filling. Under the *Tank Level Falling* category, insert the *Tank Empty* Add-on Process Trigger, as seen in Figure 21.17



**Figure 21.18: Shutdown the Outflow from the Tank and Start the Source Flow**

**Step 12:** Save and run the model. You may need to modify the speed factor to 100 to see the tank level rising.

**Question 31:** What happens when the simulation runs with regard to the tank and the two FLOWCONNECTORS?

### Part 21.3: The Gas Station

A gas station has four gas pumps, which dispense the same grade of gas. All the gas pumps pump their gas from a single tank in the ground. Cars arrive and obtain gas from any of the four pumps. If the ground tank is depleted to 100 gallons, the pumps are closed, although any car currently being filled finishes. Any cars in line will stay in line, but newly arriving cars will balk (i.e., leave). A tanker truck comes by periodically to refill the ground tank. As soon as the ground tank is full (i.e., 10,000 gallons), the pumps are open again to deliver gas.

This problem can be approached in several ways. Again, a continuous variable (i.e., tank) will be used to model the ground tank, and we will let the cars and the tanker truck change the inflow and outflow rates to the tank. The cars and the tanker truck will be our active entities. The amount of gas any car needs is assumed to be Triangular(4, 9.5, 25) gallons. The rate that any car can pump gas is Triangular(125, 150, 200) gallons per hour. The tanker will fill the tank at 20,000 gallons per hour.

<sup>308</sup> You will have to type in these state variables as they will not show up in the dropdown list of possible state variables.

**Step 1:** We will modify the model of the simple tank from the previous section. Add two SOURCES, one SERVER, three SINKS, and two MODELENTITIES named **EntCars** and **EntRefillTrucks**, as illustrated in Figure 21.19.

- **EntCars** arrive Exponentially every 1.5 minutes beginning at time 0.1, while the **EntRefillTrucks** arrive Uniformly between 6.75 and 8.25 hours apart.
- Use CONNECTORS to link all of the objects associated with the cars together. Make the link between **SrcTanker** and **SnkTankerLeaves** a PATH.<sup>309</sup>
- Make sure each of the sources creates the correct entity type. Also, change the picture of the **EntRefillTrucks** to be a tanker<sup>310</sup> and the **EntCars** to have four and six different car pictures randomly chosen (i.e., set the *Random Symbol* property to “True” under the *Animation* section).

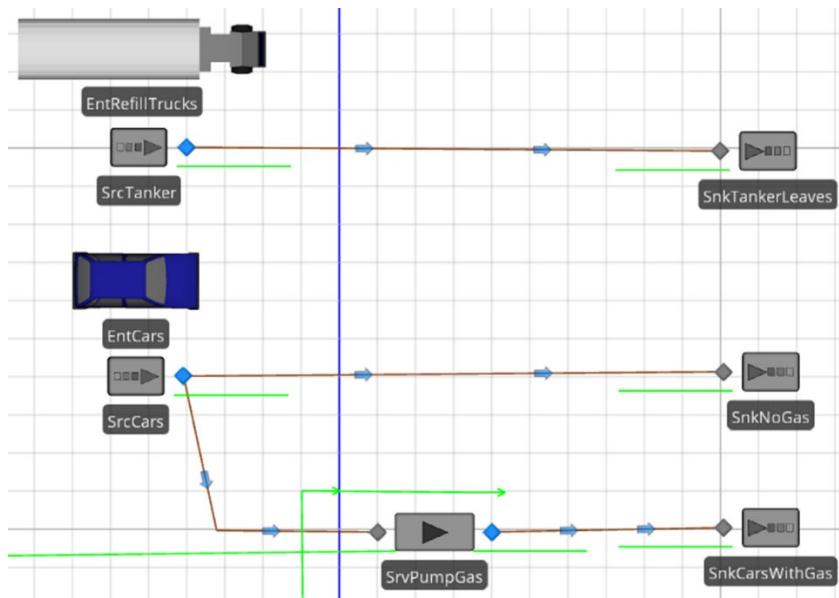


Figure 21.19: Model Structure

**Step 2:** Define a new DISCRETE INTEGER state variable for the main Model named **GstaPumpOn** to model whether the pumps are on or off. A value of one means “on,” and zero means “off.” Set the *Initial State Value* property to zero.

**Step 3:** Set the *Initial Contents* property of the **TankGas** to be **TankMin**.

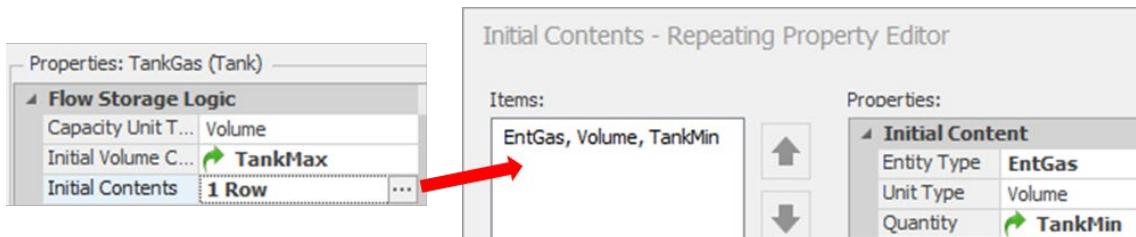
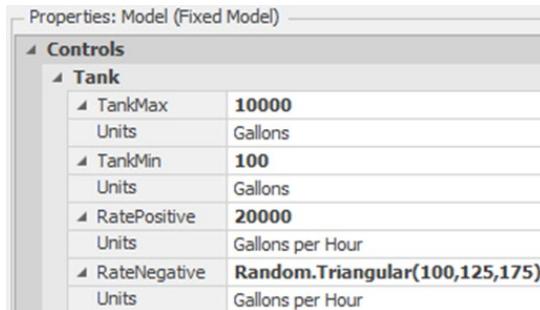


Figure 21.20: Setting the Initial Amount of Gas in the Tank

<sup>309</sup> The one path will allow the tanker be seen as it fills up the tank. Also, we have two sinks for the cars to automatically collect statistics on cars that balk and those do not.

<sup>310</sup> You might want to make the tanker a little smaller.

**Step 4:** For the **Model**, change the *TankMax* and *TankMin* properties to 10,000 gallons and 100 gallons respectively. By setting the low mark value to 100, we will not allow our tank to empty completely since cars may still be getting gas when the tanker arrives. Change the **PositiveRate** and the **RateNegative** model properties to 20,000 and `Random.Triangular(100,125,175)` gallons per hour to represent the rate a car will empty the tank.



**Figure 21.21: Setting the Rates and Initial Values of the Tank**

**Step 5:** Now, for the cars (i.e., `MODELENTITY`), let's define two state variables to represent the amount of gas needed and the rate the person can pump gas. Select the `MODELENTITY` in the [Navigation] panel. Insert the following two states, which will be characteristics of the individual cars as they arrive.

- **EStaGasNeeded** is a DISCRETE REAL STATE variable with a *Unit Type* of “Volume” and an *Initial State Value* of “0” gallons.
- **EStaCarPumpingRate** is a DISCRETE REAL STATE variable with a *Unit Type* of “VolumeFlowRate” and an *Initial State Value* of “0” gallons per hour.

**Step 6:** Since the cars arriving will set the outflow rate, we need to set the *Initial Maximum Flow Rate* of the **Output@TankGas** to “0” Gallons per hour.

**Step 7:** When cars are created by the source, define a process in the *Created Entity* add-on process trigger as in Figure 21.22, which assigns the car characteristics.<sup>311</sup>



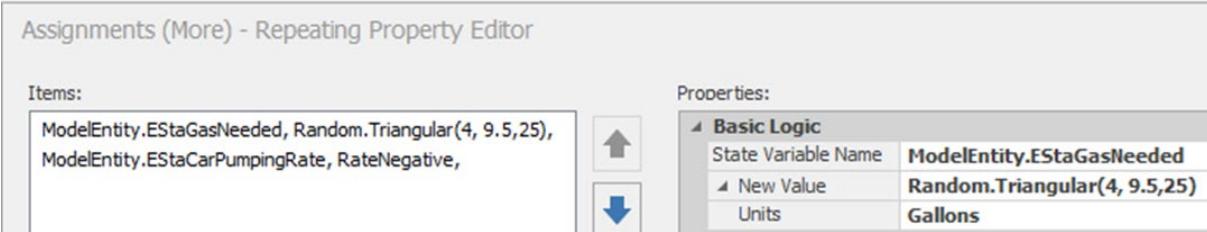
**Figure 21.22: Cars Leaving the Source**

The *Assign* step should set the following two state variables making sure the rate is negative, as seen in Figure 21.23.

```

ModelEntity.EStaGasNeeded = Random.Triangular(4, 9.5, 25)
ModelEntity.EStaCarPumpingRate = RateNegative
  
```

<sup>311</sup> Throughout this example we will use the *Assign* process step rather than using the *State Assignments* in the modeling objects. Ultimately the *Assign* steps offer more generality since the *State Assignments* are only available on entry to and exit from the object.



**Figure 21.23: Specifying Gas Needed and the Pumping Rate of Each Car**

**Step 8:** Set the *Processing Time* property (in Hours) in the **SrvPumpGas** server to be the following.

```
ModelEntity.EStaGasNeeded/ModelEntity.EStaCarPumpingRate
```

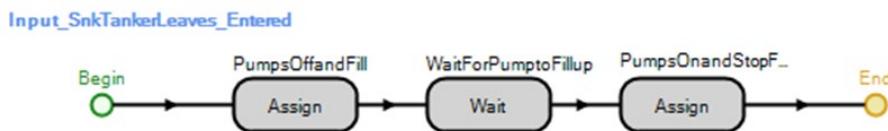
This expression takes into account the amount of gas needed divided by the rate at which the customer can pump it. Make sure to change the *Units* to “Hours” and do not use the default of “Minutes.” Be sure that the *Initial Capacity* property of the **SrvPumpGas** server is set to “4” to indicate room for four cars.<sup>312</sup>

**Step 9:** If the pumps are on, cars will be routed to the pumps and if they are off, they will be sent directly to the **SnkCarsNoGas**. The selection link weights on the paths leaving the source are defined as follows to ensure that one path will have a weight of one and the other one a zero.

- For the connection to **SnkCarsNoGas**, set the *Selection Weight* property to 1 – *GStaPumpOn*.
- For the connection to **SrvPumpGas**, the *Selection Weight* should be *GStaPumpOn*.

**Step 10:** All discrete event simulation languages use events, and the user can define their own events in SIMIO. Under the “*Definitions*” tab, add a new **EVENT** named **EvntStartPumps**, which will allow entities to wait until the pumps are started up after they have been turned off to start pumping again.

**Step 11:** Consider what happens when the tanker truck arrives, as shown in Figure 21.24. The tanker truck will turn off the pumps and start filling the tank at 20,000 gallons per hour. The tanker truck will continue to fill until the tank is full (i.e., the monitored **MonTankFull** reaches its threshold). Use the *Entered* add-on process trigger for the input node of the **SnkTankLeaves** (i.e., when the truck entity enters the sink).<sup>313</sup>



**Figure 21.24: Tanker Truck Leaves**

- In the *Assign* step, set the **GStaPumpOn** state variable to zero, turn off the flow **REGULATOR** of the **TANK**, and turn on the **REGULATOR** associated with the **FLOWSOURCE**, as seen in Figure 21.25.

<sup>312</sup> You may want to arrange the processing station animated queue to be oriented with four points around the server like a gas pump.

<sup>313</sup> It is necessary to invoke this process after the truck leaves the output at the source so that a time in system can be computed for the truck. If this process is invoked before the truck leaves the output, then its time in system is not updated.



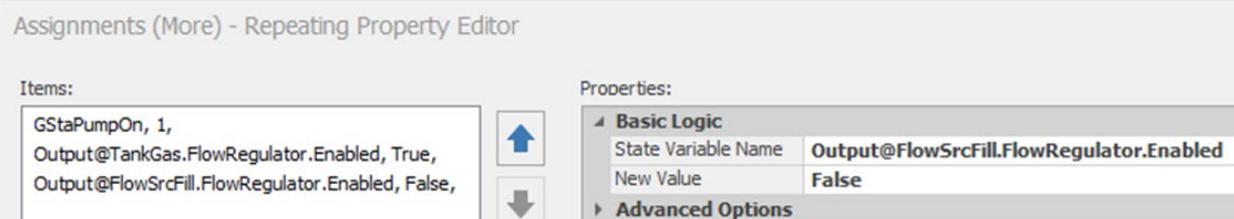
**Figure 21.25: Setting the Pump to Off and Filling the Tank**

- The *Wait* step delays the truck until the tank is filled. The full tank will be signaled by an event called **EvntStartPumps**.



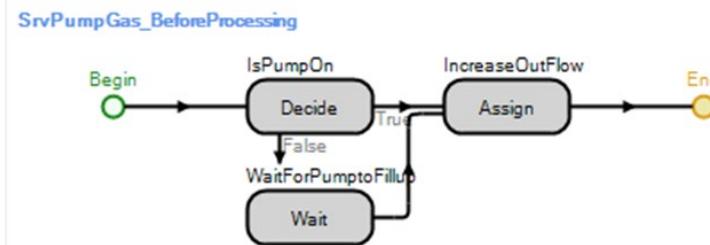
**Figure 21.26: Waiting for EvntStartPumps to Continue**

- Once the tank has filled up, we need to turn the pumps on, shut down the **FLOWSOURCE REGULATOR**, and enable the tank regulator to start again.



**Figure 21.27: Turning On Pumps and Shutdown Filling the Tanks**

**Step 12:** When the pumps are off, new cars that arrive will leave the system. However, cars that are currently in line and/or filling up will remain. Those that are currently filling up will finish, but no other car should be allowed to start filling. Therefore, within the **SrvPumpGas** server, consider first when the processing of a car filling its individual tank begins (see Figure 21.28). Insert the *Before\_Processing* Add-On Process Trigger with the following logic.

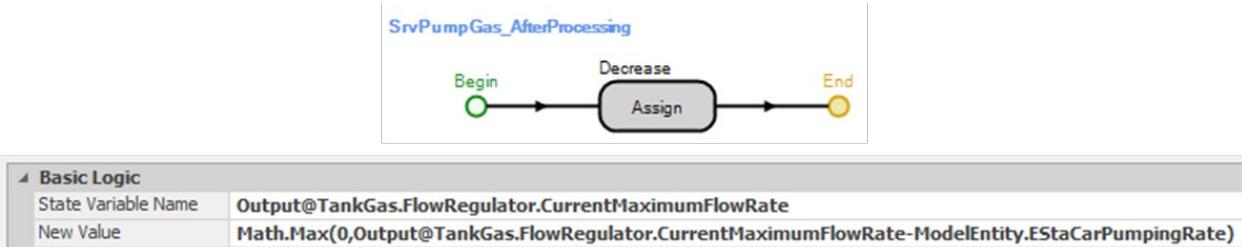


**Figure 21.28: Processing Begins at the Pump**

- When the *Before\_Processing* add-on process trigger starts: Decide if `GStaPumpOn == 1`
- If the pumps are off (i.e., “False” branch), then “Wait” for the event **EvntStartPumps**.
- Otherwise, if the pumps are on, enable the **TankGas** **FLOWREGULATOR** (i.e., `Output@TankGas.FlowRegulator.Enabled = True`) and change the tank emptying rate via an *Assign* step by adding the outflow due to the car being filled to the current rate: `Output@TankGas.FlowRegulator.CurrentMaximumFlowRate + ModelEntity.EStaCarPumpingRate`.

**Step 13:** When the car finishes (i.e., processing) at the pump (i.e., *After\_Processing* add-on process trigger for the **SrvPumpGas**), refer to Figure 21.29, we need to remove the outflow due to the car being filled (i.e., reduce rate). Use the **Math.Max** to ensure that you never go below zero.

```
Output@TankGas.FlowRegulator.CurrentMaximumFlowRate = Math.Max(0,
Output@TankGas.FlowRegulator.CurrentMaximumFlowRate -
ModelEntity.EStaCarPumpingRate)
```



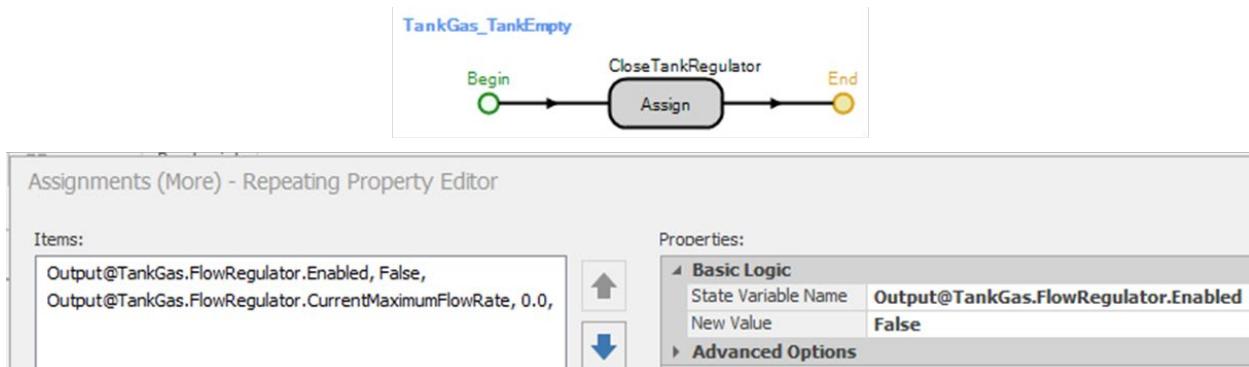
**Figure 21.29: Car Finishes at Pump Decrease Rate**

**Step 14:** Next, take care of the process when the **Tank** reaches the low mark of 100 gallons. For the **TankGas**, insert the *TankGas\_TankLevelFallingBelowLowMark* add-on process trigger and an *Assign* step that will set the **GStaPumpOn** to “0”.



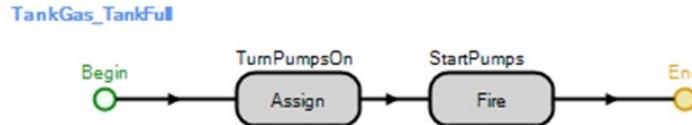
**Figure 21.30: Turn off the Pumps so Cars will Balk**

**Step 15:** For the **TankGas\_TankEmpty** (see Figure 21.32), we need to disable the tank’s outflow (i.e., set **Output@TankGas.FlowRegulator.Enabled** to “False”) and then set the rate to zero (i.e., **Output@TankGas.FlowRegulator.CurrentMaximumFlowRate=0.0**).



**Figure 21.31: Tank is Empty**

**Step 16:** For the **TankGas\_TankFull** event (see Figure 21.32), it should turn the pump on and let any cars waiting as well as the refill truck, know that the filling process has been completed. Use an *Assign* step to set the **GStaPumpOn** to “1” and then use the *Fire* step to “Fire” the **EvntStartPumps** event.



**Figure 21.32: Tank is Full**

**Step 17:** When the event **EvntStartPumps** is fired, the truck waiting for the tank to fill is activated, and the cars waiting at the pumps will be activated. If these should be activated differently, then the process needs to be modified.

**Step 18:** Save and run the current model for 48 hours.

**Question 32:** Does this system seem to work (i.e., how many cars get no gas versus those that do)?

---

**Question 33:** What happens when the tanker comes and their cars pumping gas and waiting in line?

---

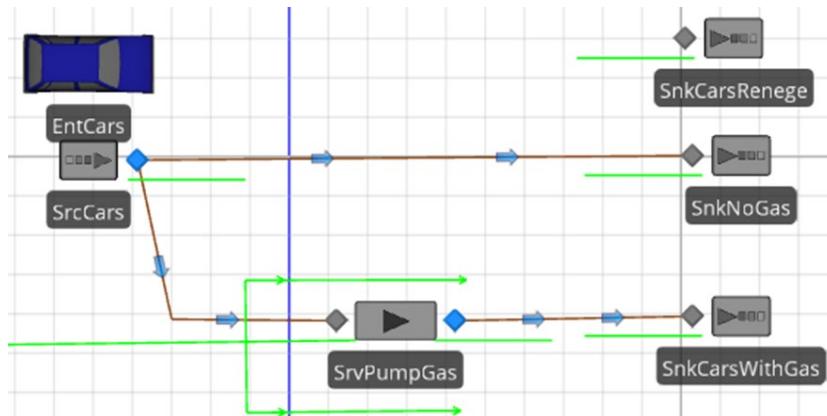
**Question 34:** What is the average level in the tank?

---

#### Part 21.4: Reneging the Cars When the Pump Goes Off

The previous model assumed that when the tanker arrived and shut the pumps off, the cars at the pumps would continue to wait until the pumps were turned back on. Let's modify the model so that all cars waiting at the pumps would renege (that is, leave).

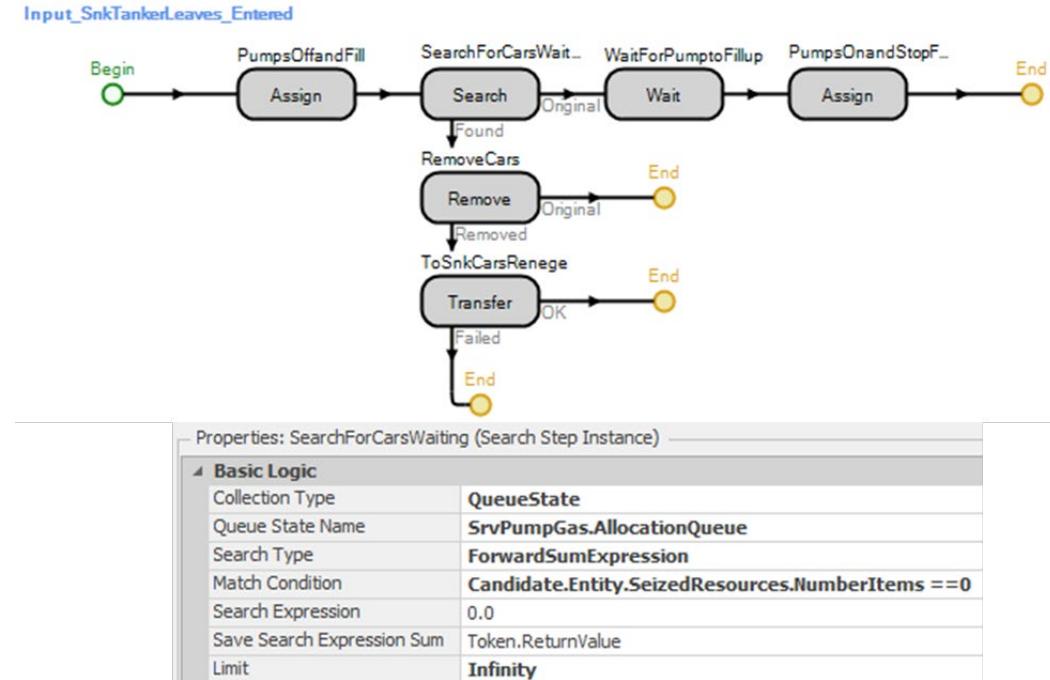
**Step 1:** Add a **SINK** to your model named **SnkCarsReneg**, which will automatically generate statistics for the renege cars but does not need to be connected.



**Figure 21.33: Adding a new Sink that is not Connected via a Path**

**Step 2:** When the tanker arrives, the cars waiting for a pump should be removed from the `input queue` of the **SrvPumpGas**. After the pump is turned off, insert a *Search* step that can search a collection of objects in a **TABLE**, a **LIST**, an instance of an object, a queue, and objects seized by the parent or associated object. In this case, set the *Collection Type* to “**QueueState**” and the *Queue State Name* to `SrvPumpGas.AllocationQueue`, which will search the cars waiting in the input buffer to be processed at the pump. See Figure 21.34.

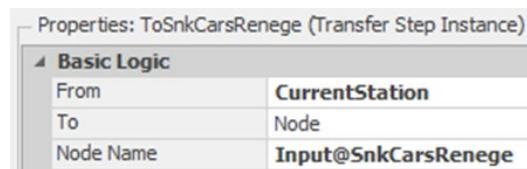
- The search needs to return all the cars waiting; therefore, set the *Limit* property to “Infinity”. The *Match Condition* will allow searching for certain objects that have to be true for the item to be selected. In this case, we could have been left blank to return all entities in the queue. However, if you chose paths to link the objects and a car was on the link and arrived at the same time the tanker arrived, then there could be a car that has seized the pump but is waiting in the “Wait” state for the pump to come on and therefore we don’t want to remove any cars that have already seized a pump. For each car (i.e., **EntCars**) found, a new TOKEN associated with the found car is sent down the “Found” path, while the original TOKEN will eventually continue the “Original” path once all items have been found<sup>314</sup>.



**Figure 21.34: Reneging all Cars Waiting for Gas when the Pump goes Off.**

**Step 3:** Insert a *Remove* step to remove the current car associated with the “Found” TOKEN from the **SrvPumpGas.AllocationQueue**. A TOKEN associated with the removed item will exit out the “Removed” path.

**Step 4:** Now insert a *Transfer* step that will move the car (i.e., **MODELENTITY**) to the **SnkCarsReneg** by specifying the *Node Name* as **Input@SnkCarsReneg** as the node to send these cars. The car is currently in the **InputBuffer** STATION, so this step needs to transfer the entity from the current station to the node specified, as seen in Figure 21.35.



**Figure 21.35: Transferring the Cars to the Renege Sink for Statistic Collection**

**Step 5:** Save and run the model, observing what happens when the Tanker arrives.

<sup>314</sup> Note that two tokens are now active.

Question 35: Do the waiting cars renege (you may need to look at the results)?

---

Question 36: How many cars reneged in 48 hours of the simulation?

---

### Part 21.5: Interrupting the Cars When the Pump Goes Off

In the previous model, the cars waiting at the pump renege when the pump was turned off, and we had already not allowed other cars to arrive at the gas pumps while the pumps were off. The cars already in service getting gas when the pumps go off were allowed to continue to fill up their car and then leave. Suppose now we want the model to interrupt the cars in service and deny them further use of the pump when the tanker arrives. These interrupted cars will leave through the **SnkCarsReneg** EXIT as well.

**Step 1:** To interrupt a process that is currently being delayed, insert an *Interrupt* step before the *Search* step, as seen in Figure 21.36 with specifications in Figure 21.37.

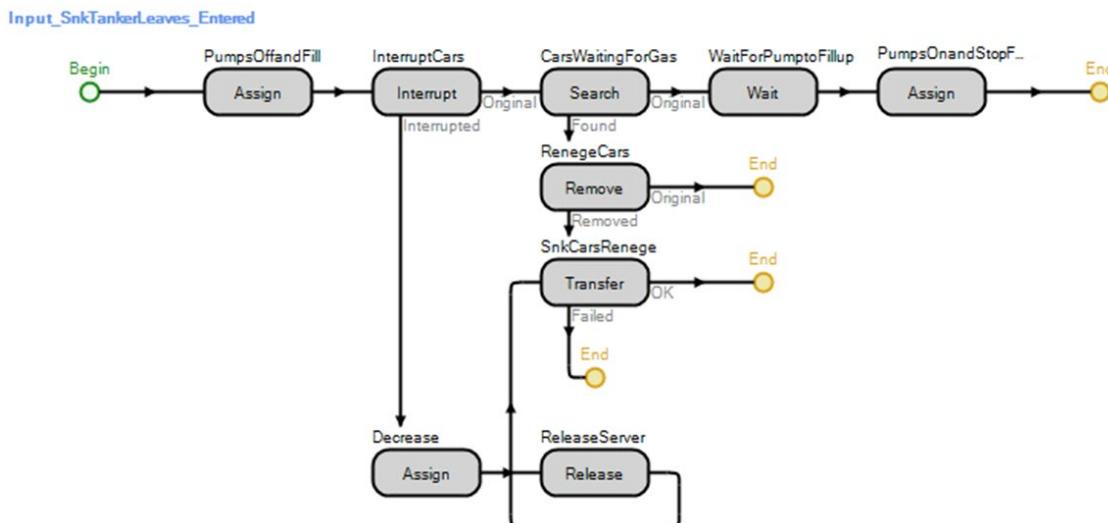


Figure 21.36: Adding the Interrupt to Force Cars Currently Processing to Reneg

- Use the *Process Name SrvPumpGas.OnEnteredProcessing* which is the process that runs the *Delay* step (i.e., processing of the cars). Notice that we “*EndProcess*” for the interrupted processes and we establish a *Limit* of “4” since there are four pumps potentially in operation.

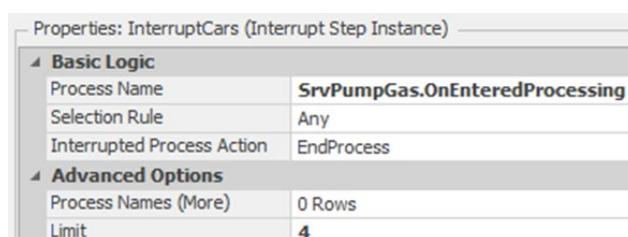


Figure 21.37: Interrupting the Cars Getting Gas

- Copy the *Assign* step from the step **SrvPumpGas \_AfterProcessing** to the “*Interrupted*” branch, which will decrease the flow rate of each car. Then, connect the *Assign* step to the *Transfer* step that was used previously to force the cars to exit the renege sink (i.e., **SnkCarsReneg**).
- Insert a *Release* step that will release a Specific resource (i.e., **SrvPumpGas**).

Resource Information	
Resource Type	Specific
Resource Name	SrvPumpGas

**Figure 21.38: Releasing the Server SrvPumpGas after Interruption**

**Step 2:** Save and run the model, observing what happens.

**Question 37:** How can you tell if the interrupt is working (Hint: put a breakpoint on Search and then single step the simulation)?

---

**Question 38:** How many cars have exited the renege sink now (i.e., how many cars were interrupted)?

---

## Part 21.6: Using Entities that Carry a Tank

In our gas scenario, there are really three distinct systems (i.e., the cars and the pump, the refill tanker, and finally, the continuous gas tank system). The first two systems are discrete systems that interact with the continuous system by turning on and off the regulators as well as changing the flow rate. The [Flow Library] provides several other flow objects that combine discrete and continuous together, as seen in Table 21.1. Currently, the tanker truck arrives and causes the FLOWSOURCE to fill up the tank at a certain rate till it is full. We assume the tanker has enough gas to fill the tank till it is full. However, the tanker truck can only carry about 4500 gallons of gas. Now, we will assume that we own one tanker that has a volume capacity (i.e., actually carries the gas) that will travel between the refill station and the gas station, filling the tank with its capacity. The tanker can move faster when it is empty on the return trip back to the refill station.

**Table 21.1: Additional Flow Objects Linking Discrete and Continuous Entities**

Flow Object	Description
CONTAINERENTITY	A MODELENTITY that contains a TANK which is transported with the entity.
FILLER	This object behaves in a similar fashion as the COMBINER. Instead of combining a parent and member entity together, the object combines (i.e., fills up the tank) a discrete container entity with a flow.
EMPTIER	This object behaves in a similar fashion as the SEPARATOR does for entities by separating (i.e., emptying the tank) the flow from the container entity.
ITEMSTOFLOWCONVERTER	Converts discrete model entities into continuous flows (e.g., large drinking bottles are delivered and placed into the water machine, which flows out water to people until the bottle is empty).
FLOWTOITEMCONVERTER	Converts continuous flows into discrete entities (i.e., continuous process of spinning of fibers to produce a bobbin of yarn which is transferred to the next station).

**Step 1:** Since we will model the truck physically transporting the gas back and forth from the refill session, we can delete things that will not be needed any longer. Delete the SnkTankerLeaves as well as the EntRefillTrucks, as we need to use a CONTAINERENTITY for the tanker now.

**Step 2:** From the Definitions tab, insert a new EXPRESSION PROPERTY named TankerCapacity with a Unit Type of “Volume” with Default Units set to “Gallons,” as seen in Figure 21.39.

Properties: TankerCapacity (Expression Property)	
Value	
Default Value	4500
Switch Property Name	
Switch Condition	Equal
Switch Value	
Candidate References	False
Unit Type	Volume
Default Units	Gallons

Figure 21.39: Properties of the TankerCapacity Property

**Step 3:** From the [Flow Library], insert a CONTANERENTITY named **CEntTankerTruck**, which has the *Initial Volume Capacity* set to the **TankerCapacity** (i.e., a 4500-gallon tanker) as seen in Figure 21.40. Change the picture to the same tanker and resize it to be smaller. Also, you may want to make the tank square and stretch it out.

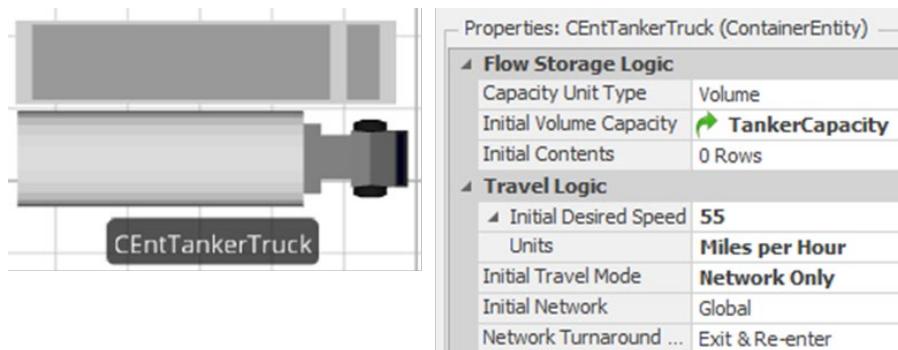


Figure 21.40: Setting up the CEntTankerTruck

**Step 4:** Select the **SrcTanker** and specify the new **CEntTankerTruck** as the *Entity Type*. Since we will only have one tanker in the system, set the *Maximum Arrivals* to “1” under the *Stopping Conditions* section.

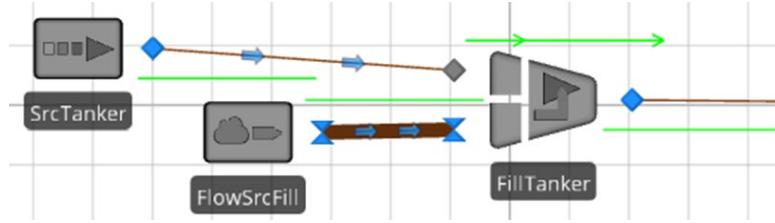
Properties: SrcTanker (Source)	
Entity Arrival Logic	
Entity Type	CEntTankerTruck
Arrival Mode	Interarrival Time
Time Offset	0.0
Interarrival Time	Random.Uniform(6.75,8.25)
Units	Hours
Entities Per Arrival	1
Stopping Conditions	
Maximum Arrivals	1

Figure 21.41: Modifying the SrcTanker to Create One Tanker

**Step 5:** Next, we will create the refill station where the tanker will be filled up and then allowed to transport the gas to the gas station, as seen in Figure 21.42.

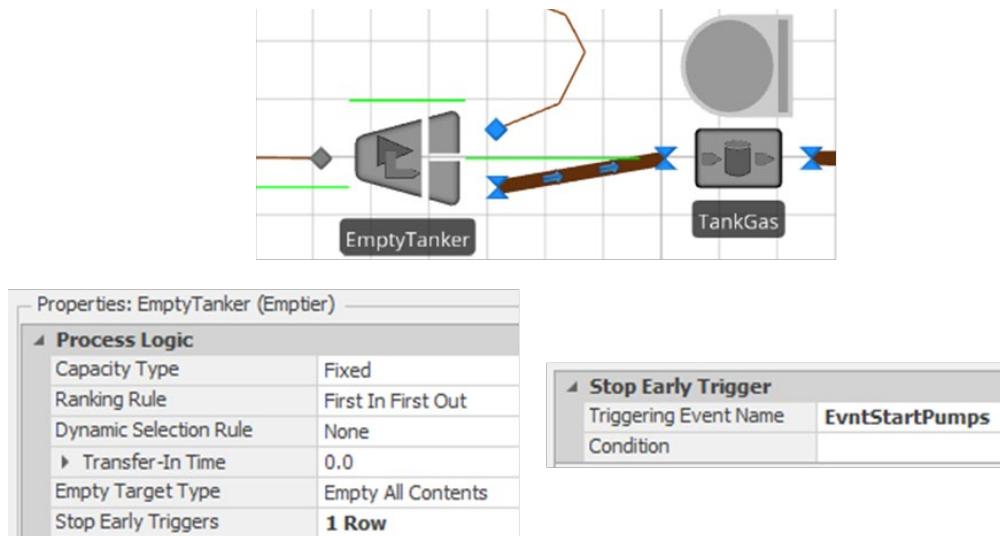
- Disconnect the **FlowSrcFill** from the **TankGas** as we will use the tanker to fill up the tank.
- Insert a new **FILLER** named **FillTanker**, which allows us to fill up a container in an entity. Also, make the processing station queue-oriented to allow the large tanker to be seen while processing.

- Connect the **SrcTanker** via a path to a **ContainerInput** node of the **FillTanker**.<sup>315</sup>
- Finally, Connect the **FlowSrcFill** to the **FlowInput** node of the **FillTanker**.



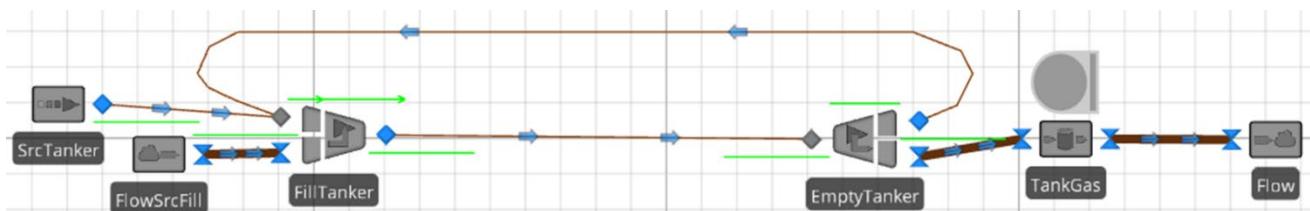
**Figure 21.42: Refill Station Area**

**Step 6:** Once the tanker arrives at the gas station, we need to pump the gas out of the tanker and into the tank. Insert a new **EMPTIER** named **EmptyTanker** and connect the **FlowOutput** node to the **TankGas** via a **FLOWCONNECTOR**. The emptier can either empty all the contents of the container entity or a specific amount. In this case, we will empty all of it until the **TankGas** is full (i.e., the **EvntStartPumps** is fired when the tank is full).



**Figure 21.43: Using the Emptier to Fill up the Gas Tank from the Tanker**

**Step 7:** Finally connect the **Output@FillTanker** to the **Input@EmptyTanker** via a time path as well as the **ContainerOutput@EmptyTanker** to the **ContainerInput@FillTanker**. Both time paths should have a Travel Time set to Random.Uniform(3.37, 4.125) hours to travel between the gas and refill station.



**Figure 21.44: Tanker and Gas Station Refilling System**

**Step 8:** Save and run the model.

<sup>315</sup> The simulation will warn you when you run the simulation and the entity that is entering the container input node does not have a tank associated with the entity (i.e., it is not a CONTAINERENTITY).

*Question 39:* Are cars being allowed to get gas immediately?

---

*Question 40:* How many trips does it take before cars are allowed into the gas station, and why?

---

*Question 41:* Does the tanker always return empty?

---

*Question 42:* How many cars renege during the 48 hours?

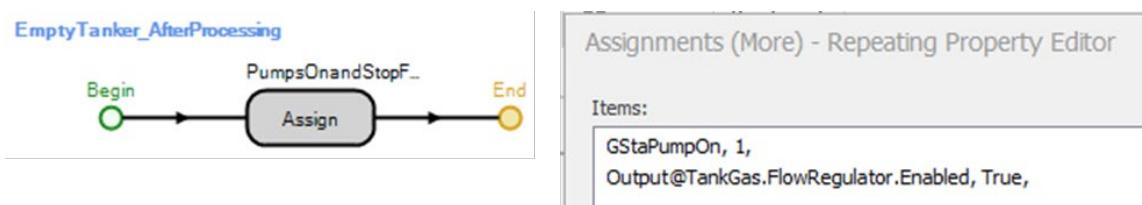
---

**Step 9:** If we want the pumps to be shut off and the cars to renege like before, in the **EmptyTanker** EMPTIER, we will need to specify the *Before Processing* add-on process trigger to be the **Input\_SnkTankerLeaves\_Entered**. Create a new *After Processing* trigger that will be used to turn the gas pumps back on, as seen in Figure 21.45.

Add-On Process Triggers	
Run Initialized	
Run Ending	
Entered	
Before Processing	<b>Input_SnkTankerLeaves_Entered</b>
Emptying Container	
Emptied Container	
After Processing	<b>EmptyTanker_AfterProcessing</b>

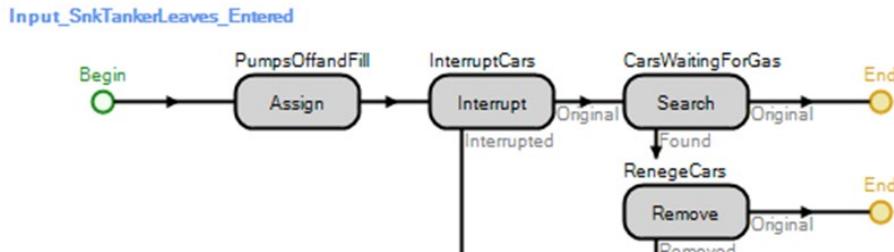
**Figure 21.45:** Using Process to Turn the Pumps On and Off when Filling up Tank

**Step 10:** If you recall, the **Input\_SnkTankerLeaves\_Entered** process shuts off the pumps and then waits for the tank to fill up before turning them back on. The **EmptyTanker** object is performing the filling task now which means we need to separate the shutting down of the pumps from the starting of the pumps. For the **EmptyTanker\_AfterProcessing**, copy the *Assign* step (i.e., “PumpsOnandStopFlow”), which is after the *Wait* step, that turns the pumps back on. You will need to delete the assignment that shuts down the FlowSrcFill output regulator since that is handled by the refilling station, as seen in Figure 21.46.



**Figure 21.46:** After the Tanker Finishes Emptying, Turn Pumps Back On

**Step 11:** For the **Input\_SnkTankerLeaves\_Entered**, delete both the Assign and Wait process steps after the *Search* step, as seen in Figure 21.47.



**Figure 21.47: Only Shutting Down the Gas Pumps and Forcing the Cars to Reneg**

**Step 12:** Save and run the model.

**Question 43:** How many cars renege now?

---

### Part 21.7: Commentary

- Continuous variables in SIMIO can be very useful, although the rates of change (second derivative) are limited to constants. The addition of the new flow objects has made it much easier to use these and integrate them into discrete systems.
- For our gas station, we could have combined the gas pumps with the continuous system and used the filler. Because the **FILLER** object can fill only one item (i.e., capacity of one) at a time, you would need a **FILLER** for each of the four gas pumps. Then, we would need to employ the concepts from Chapter 8 to restrict flow entities to the pumps (i.e., a single queue using a **RESOURCE** object). Then, the cars become container entities, which, when created, would modify their gas tank size to specify the amount of gas needed. You would connect the output of the **TankGas** to the four gas pump fillers while removing the **FLOWSSINK**.
- The uses of reneging and interruption concepts were introduced in this chapter and should not be ignored. Reneging allows entities in queues to be removed and either destroyed or transferred. Processes may also be interrupted. These features are not often used, but when they are needed, they are extremely valuable. Notice also that the modeling is accomplished through processes, not though additional specifications on the objects, as one might see in other simulation languages. This approach grants considerable flexibility to the modeler.
- The debugging facilities in SIMIO are quite extensive. When combined with animation display options, the trace, breakpoint, watch, and dashboard window can provide needed detail about the behavior of the simulation.
- The value of the “Simbits” should not be underestimated. They contain numerous examples of highly useful modeling approaches on many various topics.

# Chapter 22

## More Subclassing: Advanced Modeling of Supply Chain Systems

Chapter 10 demonstrated the modeling of a very simple three tier supply chain system. However, only the DC (Distribution Center) component in the supply chain used an inventory model to make decisions. Figure 22.1 shows a more complicated supply chain where two different store chains place orders on our manufacturer's DC. The two chain DCs run inventory models similar to the manufacturer's DC and also places replenishment orders. The manufacture can utilize two different suppliers for a particular product. The suppliers use an inventory model to replenish its inventory positions to satisfy demands from the manufacturer.

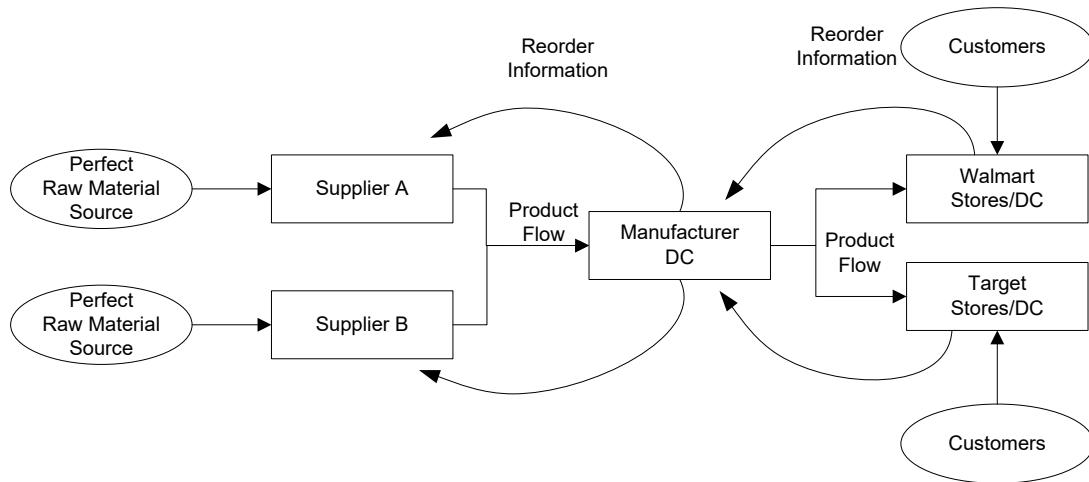


Figure 22.1: More Complicated Supply Chain

### Part 22.1: Developing a Specialized Supply Chain Workstation Object

In Chapter 10, an inventory model system was built for a single inventory system. A similar approach could be used for each segment of the supply chain in Figure 22.1 by adding additional state variables and process logic. As the system gets more complicated, changes or the requirement of different inventory models this approach is not very efficient. However, the ability to create specialized objects will make it easier to model these systems with very little additional logic.

**Step 1:** Open up the final project from Chapter10. Now also create a new project in a separate instance of SIMIO. It will be convenient to copy items from the previous model in constructing the new one.

**Step 2:** The MODELENTITY is used to model the flow of orders as well as the products. Insert the following three discrete integer state variables into the new project model by selecting the MODELENTITY object in the [Navigation] panel.

Table 22.1: Three New Discrete State Variables for Order ModelEntities

Variable	Description
EStaOrderAmt	The amount of product that is needed or has been ordered.
EStaDeliverNodeID	The input node where the order should be delivered.
EStaOriginalOrderAmt	The amount of the original order.

**Step 3:** Our goal is to encapsulate the logic and parameters that were in the simple supply solution into an object that can be used more than once. Select the **Model** from the [Navigation] panel and subclass the WORKSTATION object<sup>316</sup> and rename the new object **SUPPLYINVWORKSTATION**.

**Step 4:** Select the **SUPPLYINVWORKSTATION** object in the [Navigation] panel. Under the *Definitions→Properties*, insert the same five *Expression Standard Properties* named **InitialInventory**, **ReorderPoint**, **OrderUptoQty**, **ReviewPeriod**, and **ReviewPeriodTimeOffset**. These should all be placed in the “Inventory” Category<sup>317</sup> and the two review period properties need to have units of time with default of unit of days.

Properties			
<b>InitialInventory</b>	Expression Property	InitialInventory	Inventory
<b>ReorderPoint</b>	Expression Property	ReorderPoint	Inventory
<b>OrderUptoQty</b>	Expression Property	OrderUptoQty	Inventory
<b>ReviewPeriod</b>	Expression Property	ReviewPeriod	Inventory
<b>ReviewPeriodTimeOffset</b>	Expression Property	ReviewPeriodTimeOffset	Inventory

**Figure 22.2: Adding the Four Inventory Properties to our Supply Object**

**Step 5:** For the **SUPPLYINVWORKSTATION** object, insert a **DISCRETE INTEGER STATE** variable named **StaInventory** which will represent the current inventory amount of the product at the current segment and another one named **StaOnOrder** which represents the total amount this segment has been currently ordered from the supplier. The *Initial State Value* property should be zero for both the **StaInventory** and **StaOnOrder** variables respectively.

**Step 6:** Next, we need to insert the same **ELEMENTS** as was done for the simple supply chain model into our new object as seen in Figure 22.3 and Figure 22.4.<sup>318</sup>

- Insert/Copy two **TALLY STATISTICS** named **TallyStatSL** to track the service level performance and **TallyStatAmtOrdered** to track the amount ordered.
- Insert/Copy two **STATESTATISTICS** named **StateStatInv** and **StateStatOnOrder** to track the inventory level performance and the amount on order. Specify the *State Variable Name* property to be the **StaInventory** variable and **StaOnOrder** respectively.
- Insert a **TIMER** named **TimerReview** to model the periodic review. The supply workstation will review their inventory positions<sup>319</sup> based on the reference property **ReviewPeriod** and start reviewing based on the **ReviewPeriodTimeOffset**.

<sup>316</sup> See Chapter 18 for more information on subclassing objects in SIMIO.

<sup>317</sup> The first time, you need to type the category name “Inventory” and then the next time it can be selected from the dropdown box.

<sup>318</sup> You can copy the elements, state variables, and properties from the Chapter 10 models.

<sup>319</sup> Remember that inventory position is the inventory “on hand” plus the inventory “on order.”

Properties: StateStatInv (State Statistic Element)	
<input type="checkbox"/> Show Commonly Used Properties Only	
<input checked="" type="checkbox"/>	<b>Basic Logic</b>
State Variable Name	<b>StaInventory</b>
<input checked="" type="checkbox"/>	<b>Results Classification</b>
Data Source	<b>Inventory</b>
Category	<b>Inventory</b>
Data Item	<b>Level</b>

Properties: StateStatOnOrder (State Statistic Element)	
<input type="checkbox"/> Show Commonly Used Properties Only	
<input checked="" type="checkbox"/>	<b>Basic Logic</b>
State Variable Name	<b>StaOnOrder</b>
<input checked="" type="checkbox"/>	<b>Results Classification</b>
Data Source	<b>Inventory</b>
Category	<b>OnOrder</b>
Data Item	<b>Number</b>

Figure 22.3: Setting up the State Statistics Track Inventory and On Order

Properties: TallyStatSL (Tally Statistic Element)	Properties: TallyStatAmtOrdered (Tally Statistic Element)	Properties: TimerReview (Timer Element Instance)	
<input type="checkbox"/> Show Commonly Used Properties Only	<input type="checkbox"/> Show Commonly Used Properties Only	<input type="checkbox"/> Show Commonly Used Properties Only	
<input checked="" type="checkbox"/>	<b>Basic Logic</b>	<input checked="" type="checkbox"/>	
Unit Type	Unspecified	Unit Type	Unspecified
<input checked="" type="checkbox"/>	<b>Results Classification</b>	<input checked="" type="checkbox"/>	
Data Source	<b>Inventory</b>	Data Source	<b>Inventory</b>
Category	<b>ServiceLevel</b>	Category	<b>Amt Ordered</b>
Data Item	<b>Percentage</b>	Data Item	<b>Number</b>

Figure 22.4: Setting up the Tally Statistics and Periodic Review Timer

## Part 22.2: Adding the Ordering Station and Characteristics to Handle Orders

In the previous section, the same variables, properties, and elements were added to our new supply server object that was used for simple supply chain model of Chapter 10. Recall that the model included two separate systems (i.e., an ordering system to handle orders and one to produce the parts to fill the inventory). Figure 22.5 shows the anatomy of the new object. The original workstation will still act as the replenishment piece (i.e., convert raw material into a finished good) but it will not send the product on to the network but update the inventory level while the new order processing piece will ship out product based on customer orders.

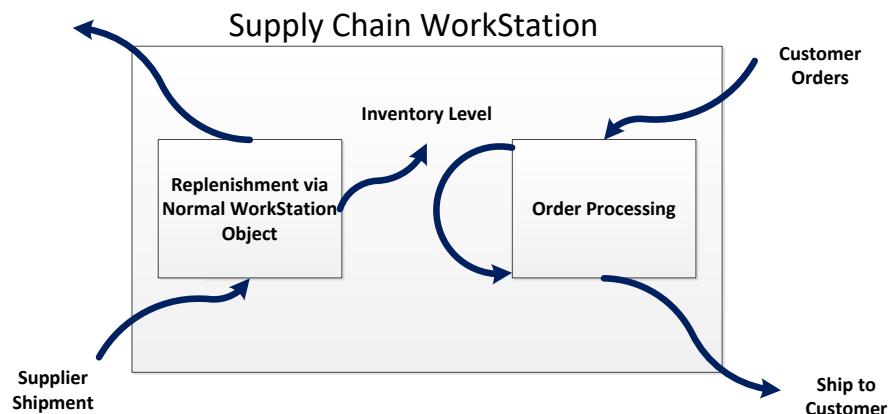
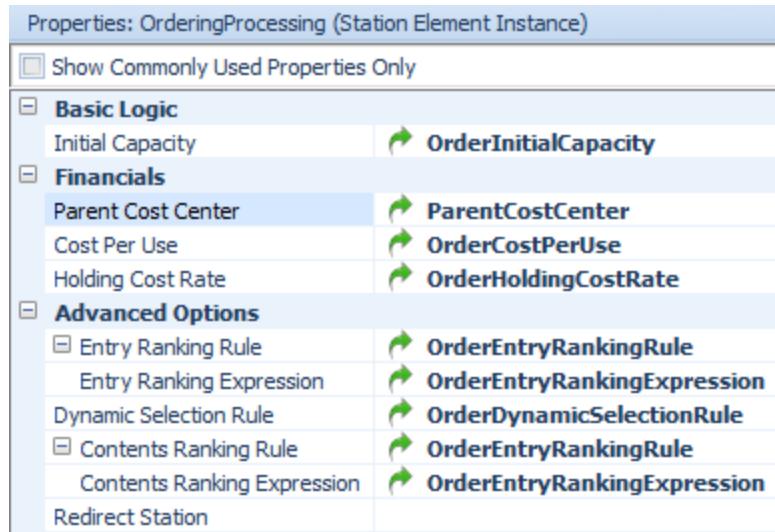


Figure 22.5: Anatomy of the Supply Chain WorkStation

**Step 1:** Our new object will need to receive orders and process them. Therefore, a location (i.e., a STATION) is needed to house these order entities while they are being processed. From the *Definitions→Elements→General* section, insert a new STATION named **OrderProcessing**.<sup>320</sup>

<sup>320</sup> See Chapter 20 for more information on STATIONS.

**Step 2:** To set capacities of the ordering process as well as sort orders waiting, create a new referenced property by right clicking on the property except the *Parent Cost Center* should be set to the **ParentCostCenter**. Use the same property names as seen in Figure 22.6.



**Figure 22.6: Creating Properties to Allow User to Change the Ranking of Orders**

**Step 3:** Switch to the *Definitions→Properties* panel and modify the new properties by placing the *OrderCostPerUse* and *OrderHoldingCostRate* into the “Financials/Order” category and the rest in the “Ordering Information” category.

OrderInitialCapacity	Expression Property	OrderInitialCapacity	Order Processing
OrderEntryRankingRule	Enumeration Property	OrderEntryRankingRule	Order Processing
OrderEntryRankingExpression	Expression Property	OrderEntryRankingExpression	Order Processing
OrderDynamicSelectionRule	Selection Rule Property	OrderDynamicSelectionRule	Order Processing
OrderCostPerUse	Expression Property	OrderCostPerUse	Financials/Order
OrderHoldingCostRate	Expression Property	OrderHoldingCostRate	Financials/Order

**Figure 22.7: All of the Properties**

**Step 4:** To separate the processing of the ordering and production systems from the “*Facility*” tab, insert a new RESOURCE named **ResOrder** which will be used in the ordering system while the workstation will be used for the production system. Also, the resource should not be visible externally in the model.<sup>321</sup>

**Step 5:** Next, we will link the new resource capacity and ranking to the ordering station. For the *Capacity Type* and *Work Schedule* properties create new reference properties, while for the others just choose the ones already created.

---

<sup>321</sup> To make sure an object is not visible in the external view, right click the object and toggle the “Externally Visible” button.

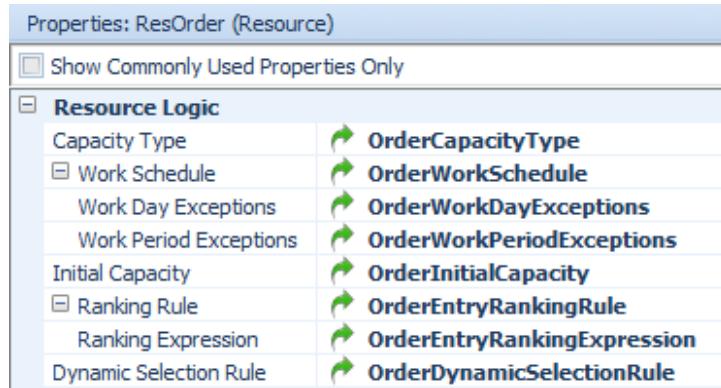


Figure 22.8: Process Logic Properties of the New Resource

**Step 6:** Also, make sure to change the category of the new properties to “Ordering Information” as well as move the two new properties up above the **OrderInitialCapacity** as seen in Figure 22.9. Let’s create a Switch Condition on the new **OrderWorkSchedule** and **OrderInitialCapacity** which will not show it unless the user selects a “WorkSchedule” as the capacity type or “Fixed” respectively. This will allow the ordering processing system to follow work schedules. Do the same for work day and work period exceptions.

Properties: OrderWorkSchedule (Schedule Property)		Properties: OrderInitialCapacity (Expression Property)	
<input type="checkbox"/> Show Commonly Used Properties Only		<input type="checkbox"/> Show Commonly Used Properties Only	
<b>Logic</b>		<b>Logic</b>	
Default Value		Default Value	<b>Infinity</b>
Switch Property Name	<b>OrderCapacityType</b>	Switch Property Name	<b>OrderCapacityType</b>
Switch Condition	Equal	Switch Condition	Equal
Switch Value	<b>WorkSchedule</b>	Switch Value	<b>Fixed</b>
Schedule Type	<b>CapacitySchedule</b>	Candidate References	False
<b>Appearance</b>		<b>Appearance</b>	
Display Name	<b>OrderWorkSchedule</b>	Display Name	<b>OrderInitialCapacity</b>
Category Name	<b>Order Processing</b>	Category Name	<b>Order Processing</b>
Category Expanded	False	Category Expanded	False
Parent Property Name		Parent Property Name	
<b>General</b>		<b>General</b>	
Name	<b>OrderWorkSchedule</b>	Name	<b>OrderInitialCapacity</b>
Description		Description	
Required Value	<b>False</b>	Required Value	True
Visible	True	Visible	

Figure 22.9: Modifying the Order Capacity Type and OrderWorkSchedule Properties

**Step 7:** Next, we need to add an entry point (i.e., input node) for orders coming into the supply inventory server as well as an exit point for reorders leaving the supply server. For the **SUPPLYINVWORKSTATION** object, navigate to the *Definitions*→*External* section. Let’s add a Status Label with an expression “Sup Inv” as on top of the picture as seen in Figure 22.10 to distinguish from the original **WORKSTATION** object.

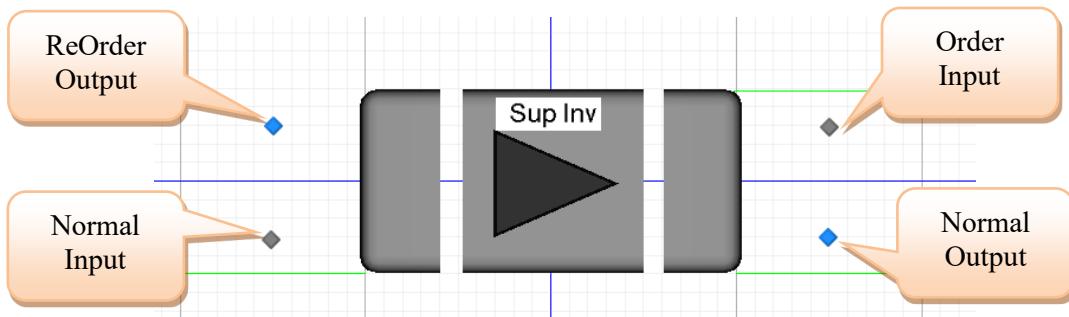


Figure 22.10: Creating the External Interface Adding Entry and Exit Nodes

**Step 8:** Now, move the original two “External” nodes (**Input** and **Output**) to the bottom third of the object as seen in Figure 22.10. Insert two additional “External” nodes<sup>322</sup> named **ReorderOutput** and **OrderInput**. The **ReorderOutput** node is a **TRANSFERNODE** and should be right above the normal **Input** node since orders will be transferred out to their supplier while the **OrderInput** node is a **BASICNODE** placed right above the normal **Output** node which will accept arriving orders. This will facilitate the notion of products flowing from the back of the chain to the front of the chain (i.e., left to right) while orders will flow the opposite direction. The properties for both nodes are specified in Figure 22.11. Once order entities enter the **OrderInput** they will be transferred directly to the **OrderProcessing** STATION.

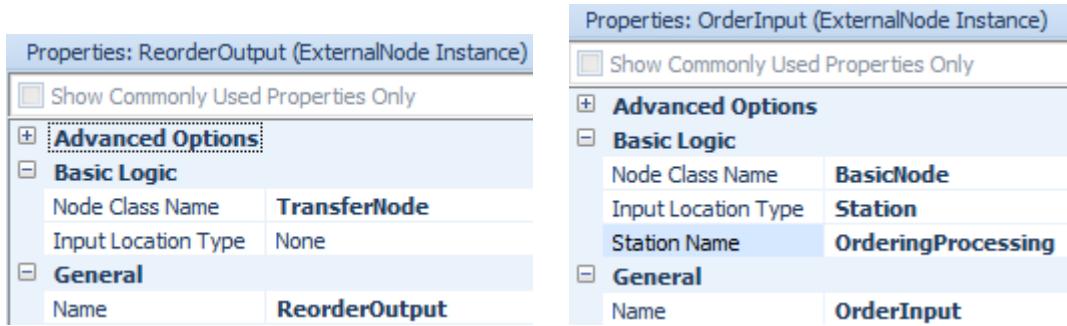


Figure 22.11: Specifying the Properties of the ReorderOutput and OrderInput Nodes

**Step 9:** Next add an additional animation queue associated with the contents queue of the OrderProcessing station as seen in Figure 22.12.<sup>323</sup> Add the same six status labels as before which represent the inventory level (**StaInventory**), average service inventory (**StateStatInv.Average**), and average service level (**TallyStatSL.Average**).<sup>324</sup>

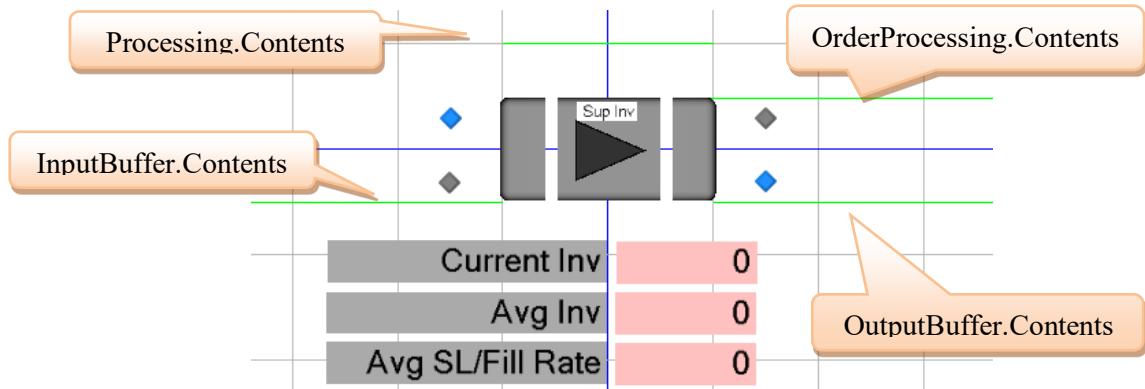


Figure 22.12: Adding the Animation Queues and Status Labels

**Step 10:** As part of the ordering system, three additional properties will be needed, as seen in Table 22.2 below. Also, the **OrderProcessingTime** property should have units of “Time” with the default units being “Days” placing these properties in the “Ordering Information” category.

<sup>322</sup> See Chapter 17 and Chapter 20 for more information on inserting external nodes into the external view of the object.

<sup>323</sup> For each queue, set the *Alignment* to “None” under the *Appearance* tab.

<sup>324</sup> Just copy the six labels from the Chapter 10 model that is currently open to get all the expressions correct.

**Table 22.2: Three Properties to Allow for the Ordering Process to Take Place**

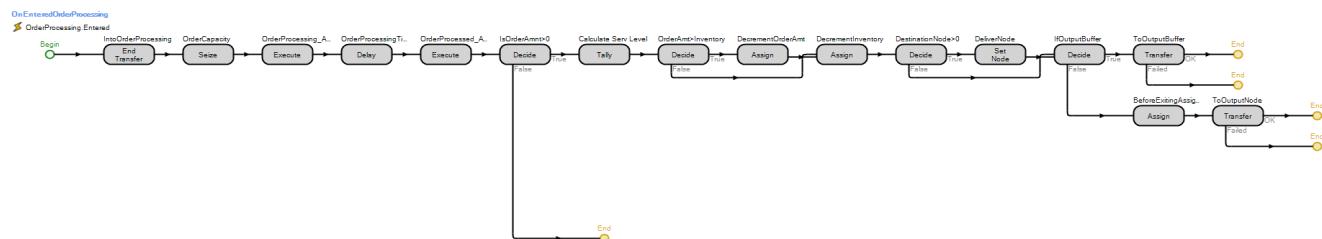
Property	Type	Description	Required
<b>OrderProcessingTime</b>	<i>Standard Property→Expression</i>	The time spent processing order before it is filled	Yes
<b>DeliverNodeID</b>	<i>Standard Property→Expression</i>	The input node id where the order should be delivered	Yes
<b>OrderEntityType</b>	<i>Object Reference Standard Property→Entity</i>	The type of order entity to send out	Yes

The *OrderEntityType* will be used in a similar fashion as the *EntityType* property for the **SOURCE** object. This allows the object to create Reorders based on a user defined type (i.e., allows the modeler to make more changes to the orders without having to modify the **SUPPLYINVWORKSTATION** object).

### Part 22.3: Adding the Behavior Logic for the Ordering System

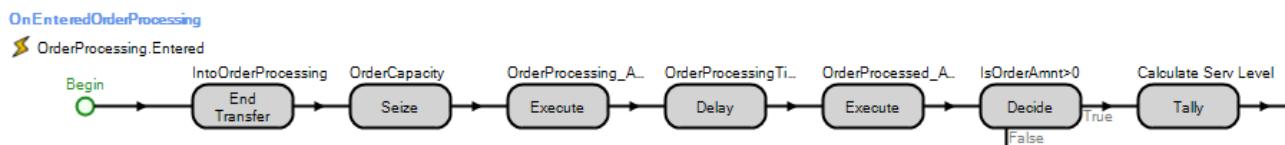
We now have defined all of the characteristics of the supply inventory object and now we need to add the ordering system. Orders that arrive to the **OrderInput** node are sent directly to the **OrderProcessing** STATION. At this point, the logic necessary to fill the incoming orders needs to be added. Recall Chapter 20 on the **DELAY** object, STATIONS respond to entities arriving and exiting the particular station.

**Step 1:** Insert a new process named *OnEnteredOrdering* with the triggering event being *OrderProcessing.Entered* as seen in Figure 22.13 to handle the processing of orders. The next few steps will be broken down into sections in order to build the process.



**Figure 22.13: The *OnEnteredOrdering* Process**

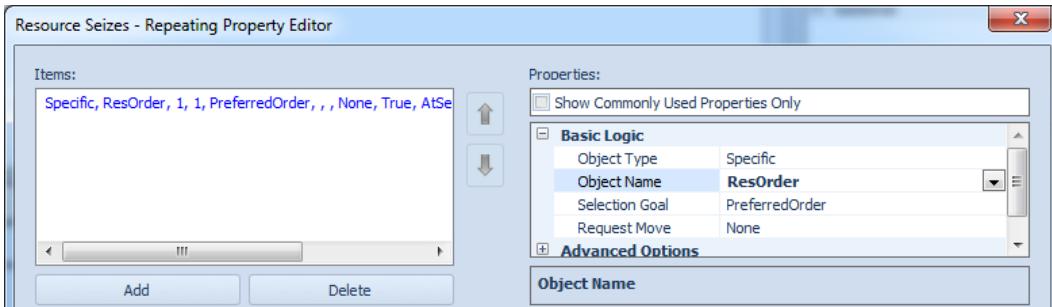
**Step 2:** When entities are transferred from one location to another, the transfer has to be ended to signal to the transferring step that it has reached its destination. Recall the **OrderProcessing** STATION's capacity was set as a reference property to allow for capacitating the order processes. Also, we added an order processing expression that would simulate the processing of the order (i.e., checking information, credit, etc.,) before it can be filled. We will only try to fulfill the order if the order amount is greater than zero and at that point we will update the service level statistics. Figure 22.14 shows the first few steps of the process while Figure 22.18 shows the properties associated with the process steps.



**Figure 22.14: First Five Steps of Handling the Arriving of Orders**

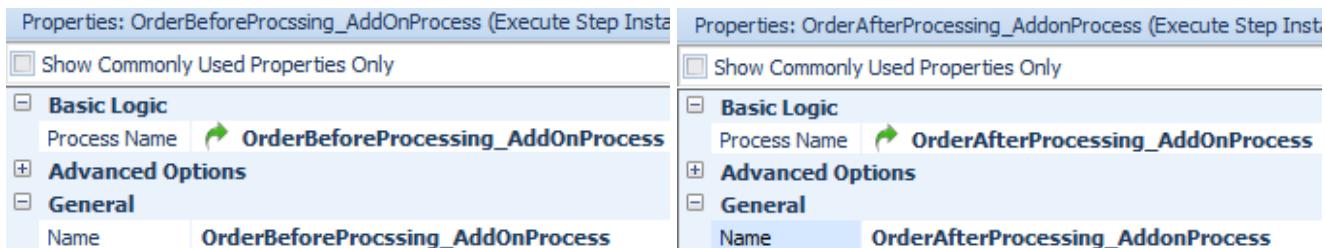
- Insert an *End Transfer* step to end the transfer of the arriving order entity from the **OrderInput** external node into the **OrderProcessing** STATION.

- Insert a *Seize* step that will seize one unit of capacity from the “Specific” resource which is the **ResOrder** as shown in Figure 22.15.



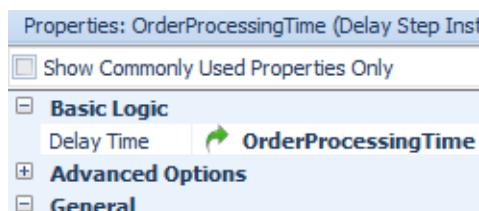
**Figure 22.15: Seizing a Unit of ResOrder Capacity**

- Like the normal WORKSTATION we would like to provide the capability for the user to execute add-on process triggers before processing the order and after the order has been processed, insert two *Execute* steps. Create a new reference property named **OrderBeforeProcessingAddOn** and **OrderAfterProcessingAddOn** as the *Process Name* property as seen in Figure 22.16. In the *Definitions→Property* section, change the category of this new reference property to “Add on Process Trigger” and the display name to “Order Processing” and “Order Processed” respectively.



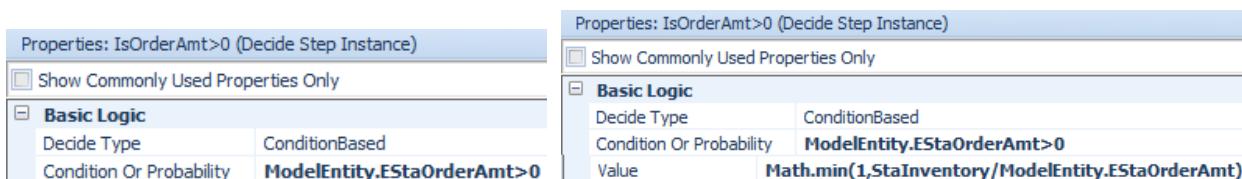
**Figure 22.16: Order Processing and Processed Add-on Process Trigger**

- Next, insert a *Delay* step between the two *Execute* statements to represent the processing of the order before filling the order using the existing *OrderProcessingTime* property.



**Figure 22.17: Setting up the *Delay* Step**

- Add a *Decide* step to determine if the order amount is greater than zero before continuing on to the fulfillment process.
- Update the service level statistic using a *Tally* step using the same logic from Chapter 10.



**Figure 22.18: Properties Associates with Beginning of the Order Process**

**Step 3:** The previous steps were similar to the ones used in the simple supply chain model in Chapter 10. These were the only ones associated with the ordering process since the DC was the object of interest and orders did not flow to the downstream customer in the supply chain. The actual filling and shipping of the order to the customer will be handled by the last steps of the process as seen in Figure 22.19.

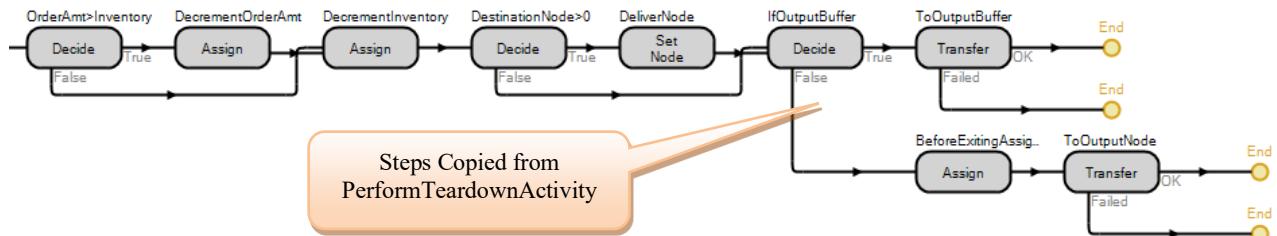


Figure 22.19: Steps Used to Fill the Order and Ship it out to the Next Element

**Step 4:** The first section (i.e., the first three steps) will update the inventory position by the amount being shipped out. If the current inventory position cannot handle the order amount (i.e., *Decide* step) then set the order amount equal to the current inventory level.<sup>325</sup> If the inventory is large enough (i.e., “*False*” branch) go directly to decrementing the current inventory position by the order amount. Insert the *Decide* and the two *Assign* steps with the properties in Figure 22.20. Otherwise, we need to change the order amount of the entity to let the customer know how much we were able to ship to them.

Properties: OrderAmt>Inventory (Decide Step Instance)	Properties: ChangeOrderAmt (Assign Step Instance)								
<input type="checkbox"/> Show Commonly Used Properties Only	<input type="checkbox"/> Show Commonly Used Properties Only								
<input checked="" type="checkbox"/> <b>Basic Logic</b> <table> <tr> <td>Decide Type</td><td>ConditionBased</td></tr> <tr> <td>Condition Or Probability</td><td><b>ModelEntity.EStaOrderAmt&gt;StaInventory</b></td></tr> </table>	Decide Type	ConditionBased	Condition Or Probability	<b>ModelEntity.EStaOrderAmt&gt;StaInventory</b>	<input checked="" type="checkbox"/> <b>Basic Logic</b> <table> <tr> <td>State Variable Name</td><td><b>ModelEntity.EStaOrderAmt</b></td></tr> <tr> <td>New Value</td><td><b>StaInventory</b></td></tr> </table>	State Variable Name	<b>ModelEntity.EStaOrderAmt</b>	New Value	<b>StaInventory</b>
Decide Type	ConditionBased								
Condition Or Probability	<b>ModelEntity.EStaOrderAmt&gt;StaInventory</b>								
State Variable Name	<b>ModelEntity.EStaOrderAmt</b>								
New Value	<b>StaInventory</b>								
Properties: Decrement Inv Level (Assign Step Instance)	Properties: Decrement Inv Level (Assign Step Instance)								
<input type="checkbox"/> Show Commonly Used Properties Only	<input type="checkbox"/> Show Commonly Used Properties Only								
<input checked="" type="checkbox"/> <b>Basic Logic</b> <table> <tr> <td>State Variable Name</td><td><b>StaInventory</b></td></tr> <tr> <td>New Value</td><td><b>Math.max(0,StaInventory-ModelEntity.EStaOrderAmt)</b></td></tr> </table>	State Variable Name	<b>StaInventory</b>	New Value	<b>Math.max(0,StaInventory-ModelEntity.EStaOrderAmt)</b>	<input checked="" type="checkbox"/> <b>Basic Logic</b> <table> <tr> <td>State Variable Name</td><td><b>StaInventory</b></td></tr> <tr> <td>New Value</td><td><b>Math.max(0,StaInventory-ModelEntity.EStaOrderAmt)</b></td></tr> </table>	State Variable Name	<b>StaInventory</b>	New Value	<b>Math.max(0,StaInventory-ModelEntity.EStaOrderAmt)</b>
State Variable Name	<b>StaInventory</b>								
New Value	<b>Math.max(0,StaInventory-ModelEntity.EStaOrderAmt)</b>								
State Variable Name	<b>StaInventory</b>								
New Value	<b>Math.max(0,StaInventory-ModelEntity.EStaOrderAmt)</b>								

Figure 22.20: Updating the Inventory Position

**Step 5:** After the inventory position has been updated, the shipment has to be sent out the normal **Output** external node. If the deliver node “id” was specified (i.e., `ModelEntity.EStaDeliverNodeID > 0`) in the `MODELENTITY` then we will set the destination node before transferring the shipment out. Otherwise just ship it out, and let the links handle where the shipment will go (see Figure 22.21).

Properties: SpecificDeliverNode (Decide Step Instance)	Properties: DeliverNode (SetNode Step Instance)								
<input type="checkbox"/> Show Commonly Used Properties Only	<input type="checkbox"/> Show Commonly Used Properties Only								
<input checked="" type="checkbox"/> <b>Basic Logic</b> <table> <tr> <td>Decide Type</td><td>ConditionBased</td></tr> <tr> <td>Condition Or Probability</td><td><b>ModelEntity.EStaDeliverNodeID&gt;0</b></td></tr> </table>	Decide Type	ConditionBased	Condition Or Probability	<b>ModelEntity.EStaDeliverNodeID&gt;0</b>	<input checked="" type="checkbox"/> <b>Basic Logic</b> <table> <tr> <td>Destination Type</td><td><b>IDNumber</b></td></tr> <tr> <td>Node ID Number</td><td><b>ModelEntity.EStaDeliverNodeID</b></td></tr> </table>	Destination Type	<b>IDNumber</b>	Node ID Number	<b>ModelEntity.EStaDeliverNodeID</b>
Decide Type	ConditionBased								
Condition Or Probability	<b>ModelEntity.EStaDeliverNodeID&gt;0</b>								
Destination Type	<b>IDNumber</b>								
Node ID Number	<b>ModelEntity.EStaDeliverNodeID</b>								

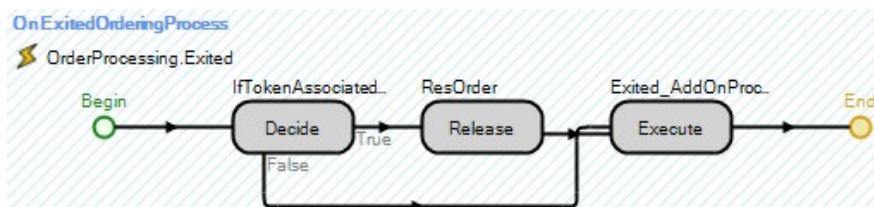
Figure 22.21: Shipping the Order Out to Downstream Customer

<sup>325</sup> Before we just set the inventory level to 0 if the order amount was greater than zero but we are now shipping it out.

**Step 6:** If the output buffer capacity is greater than zero, we will transfer it to the output buffer, otherwise the order is sent out automatically. As shown in Figure 22.19, copy the *Decide*, *Assign* and two *Transfer* steps from the end of the “*PerformTeardownActivity*” process since it is the same (see Figure 22.28).

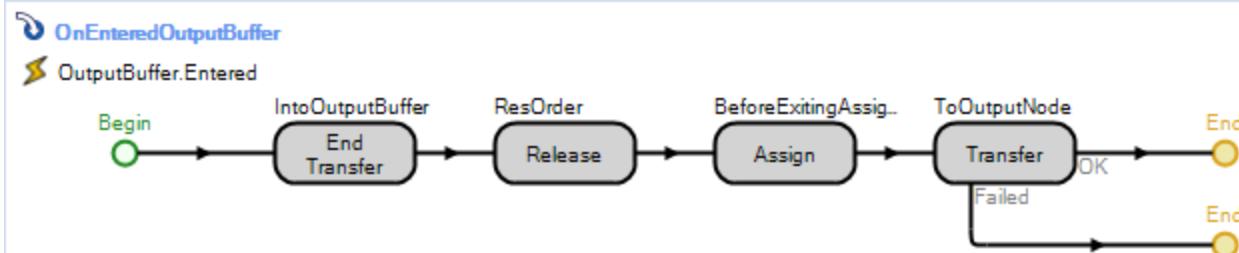
**Step 7:** Next, link the “False” branch from the *Decide* on whether they specified a deliver node to the *Decide* of the output buffer greater than zero.

**Step 8:** Insert a new process named “*OnExitedOrderingProcess*” with the triggering event being *OrderProcessing.Exited* as seen in Figure 22.22 which it executes the “*ExitedAddOn*” process trigger if it is specified. Copy all three steps from the “*OnExitedProcessing*” process. This process will run and execute all the steps when the output buffer capacity is zero and the orders are directly sent to the external node. Modify the *Release* step to release the “*Specific*” resource **ResOrder** rather than the capacity of the parent object, changing the “*Units per Object*” to 1.



**Figure 22.22: Process that Executes when Entities leave the OrderProcessing STATION.**

**Step 9:** Since the orders are being shipped out through the output buffer, we need to override the “*OnEnteredOutputBuffer*” so it releases the **ResOrder** rather than the parent object like the previous step. Click on the process and override and modify the *Release* step to release one unit of **ResOrder**.

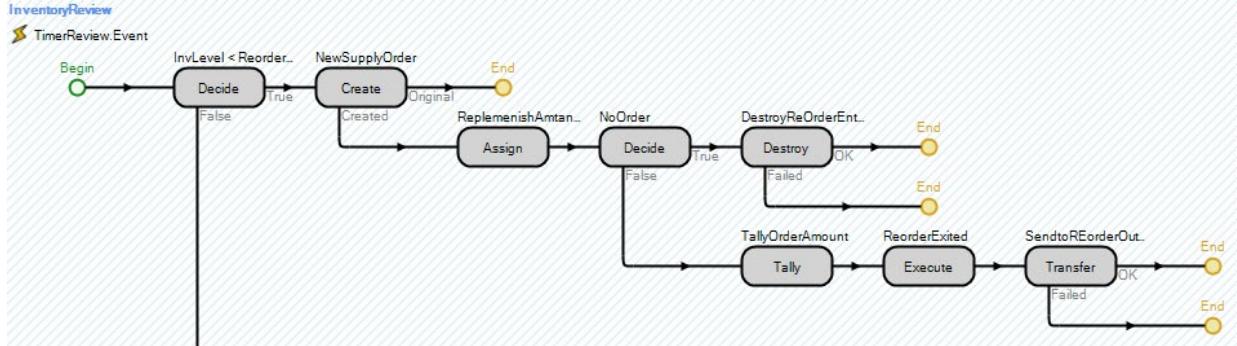


**Figure 22.23: Modifying the OutputBuffer Entered Process**

## Part 22.4: Adding the Behavior Logic for the Inventory Replenishment System

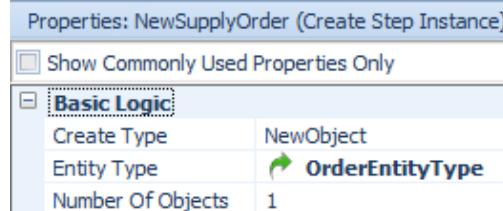
We now have defined the logic for the ordering and fulfillment system. Next, the logic necessary to implement the inventory replenishment system will be added to the new object.

**Step 1:** Setup the periodic review by creating a process to respond to the timer **TimerReview**. From the “*Processes*” tab of the **SUPPLYINVWORKSTATION** object, create a new process named **InventoryReview**. Set the *Triggering Event* property to respond to the *TimerReview.Event* which causes the process to be executed every time the timer event fires, as seen in Figure 22.24. This is almost identical to the same process created in Chapter 10. Copy the entire process and all steps can be just copied and then the *Create*, *Assign* and *Transfer* steps need to be modified. You will need to modify all the GSta variables to Sta since they are part of the workstation.



**Figure 22.24: Process to Reorder**

- The *Decide* step does a “ConditionBased” check to see if the current inventory is less than the reorder point (i.e., `StaInventory < ReorderPoint`)
- If a new reorder is needed, use the *Create* step to create a new entity as seen in Figure 22.25 which creates the entity based on the *OrderEntityType* reference property this time.



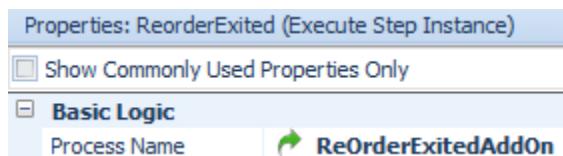
**Figure 22.25: Creating a New Reorder to Send the Supplier**

- For the newly created order, use an *Assign* step to assign the order amount and then increase the **WIP** value based on the order amount. Additionally, we will also set the original amount to help model reliability of the suppliers as well as the deliver node, if it has been specified, as in Table 22.3.

**Table 22.3: Determining the Order Amount and Updating the WIP**

State Variable	New Value
<code>ModelEntity.EStaOrderAmt</code>	<code>Math.Max(0, OrderUpToQty - StaInventory - StaOnOrder)</code>
<code>StaOnOrder</code>	<code>StaOnOrder + ModelEntity.EStaOrderAmt</code>
<code>ModelEntity.EStaDeliverNodeID</code>	<code>DeliverNodeID</code> (This is a property)
<code>ModelEntity.EStaOriginalOrderAmt</code>	<code>ModelEntity.EStaOrderAmt</code>

- Insert an *Execute* step that will run a user defined add-on process trigger which will be executed before the reorder exits the system to be sent to the supplier (see Figure 22.26). In the *Definitions→Property* section, change the category of this new reference property to “Add-On Process Triggers” and the display name to “ReOrder Exited.”



**Figure 22.26: Inserting a User-Defined Add-on Process Trigger**

- Once the *OrderEntityType* has been set, it needs to be sent to the supplier by using a *Transfer* step to move it from “FreeSpace” to the **ParentExternalNode** of the **ReOrderOutput**.<sup>326</sup>

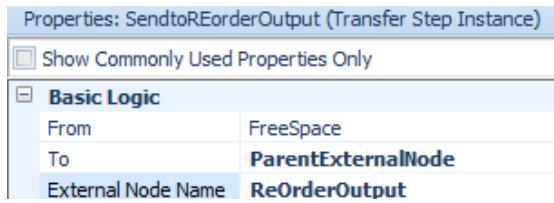


Figure 22.27: Sending the New Reorder to the Supplier

**Step 2:** Recall from Chapter 10 when the products arrived back at the DC and had been packaged, the **StaInventory** variable was increased by the order amount, while the **StaOnOrder** variable was reduced by the value in the “Processed” add-on process trigger for the **WrkDC**. Since we are encapsulating this same logic inside our new **SUPPLYINVWORKSTATION** object, the same logic needs to be added right after the “Finished Good Operation” trigger executed in the *PerformTeardownActivity* process. This process is run when the batch of parts has finished processing and the tear down activity has finished for the batch. Select the *PerformTeardownActivity* process from the “Processes” tab and then click the override button (  ) which allows you to modify it as seen in Figure 22.28.<sup>327</sup>

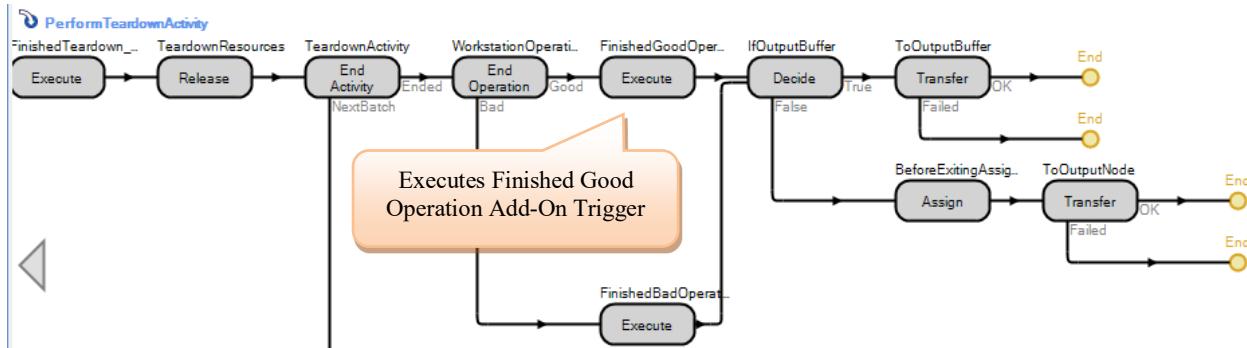
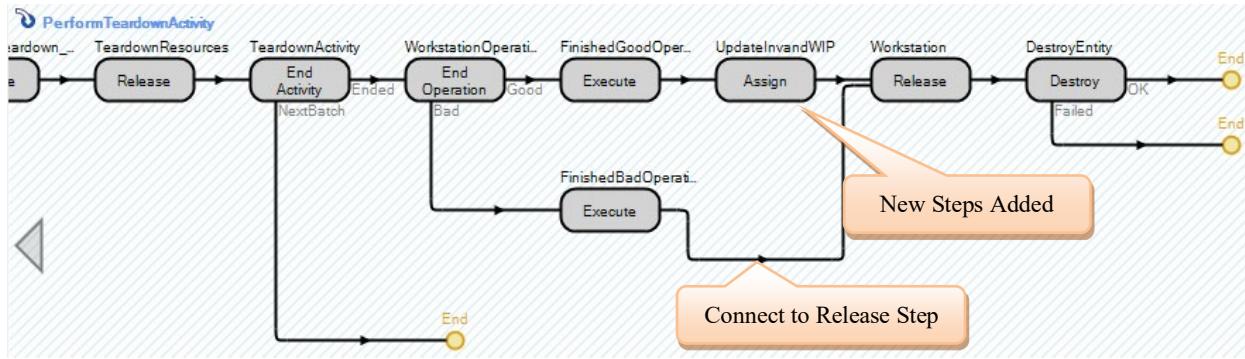


Figure 22.28: The Default *OnEnteredProcessing* Process

**Step 3:** After the execution of the “Finished Good Operation” add-on process trigger as seen in Figure 22.28, the entity will be transferred to the output buffer if the buffer capacity is greater than zero (i.e., “True” branch) or it will execute any before exiting assignments and then transferred to the external output node. In our make-to-stock model, the product is placed into inventory and not sent on to the next process. Therefore, you need delete all the steps after the *Execute* step and then insert three new process steps (i.e., *Assign*, *Release*, and *Destroy*) as seen in Figure 22.29.

<sup>326</sup> Now the modeler will indicate where the reorders should travel via a link from the **ReOrderOutput** node.

<sup>327</sup> See Chapter 19 and Chapter 20 for more information on overriding and restoring processes.



**Figure 22.29: The New *PerformTeardownActivity* Process**

- The *Assign* step updates the **Inventory** and **WIP** values as specified in Table 22.4.

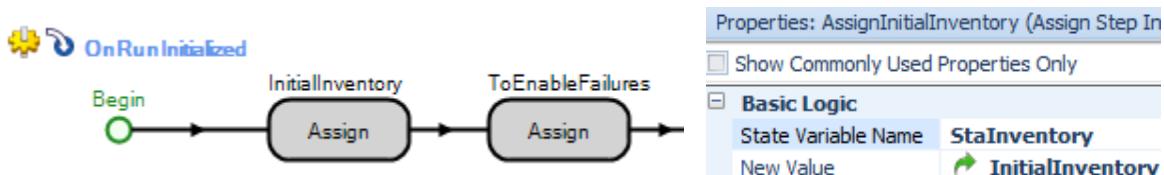
**Table 22.4: Updating Inventory and DC once Product Arrives**

State Variable	New Value
StaInventory	StaInventory + ModelEntity.EStaOrderAmt
StaOnOrder	StaOnOrder - ModelEntity.EStaOriginalOrderAmt

- Since the entity is no longer sent to the Output Buffer (i.e., it will not enter the *OnEnteredOutputBuffer* process) or to the external node which will not run the *ExitedProcessing* process, the workstation capacity is never released. Therefore, copy the *Release* step from the *OnExitedProcessing* process after the *Assign* step and make sure *Exclusion Property* is empty (i.e. reset).
- Insert a *Destroy* step to destroy the entity order that has arrived since it will not continue.
- Make sure to connect the end of the *FinishedBadOperation* add on process execution step to the *Release* step.

Recall, the initial inventory property was used to set the beginning **StaInventory** position in the *OnRunInitialized* process. Override the *OnRunInitialized* process of our new object and insert the same *Assign* step as before as seen in

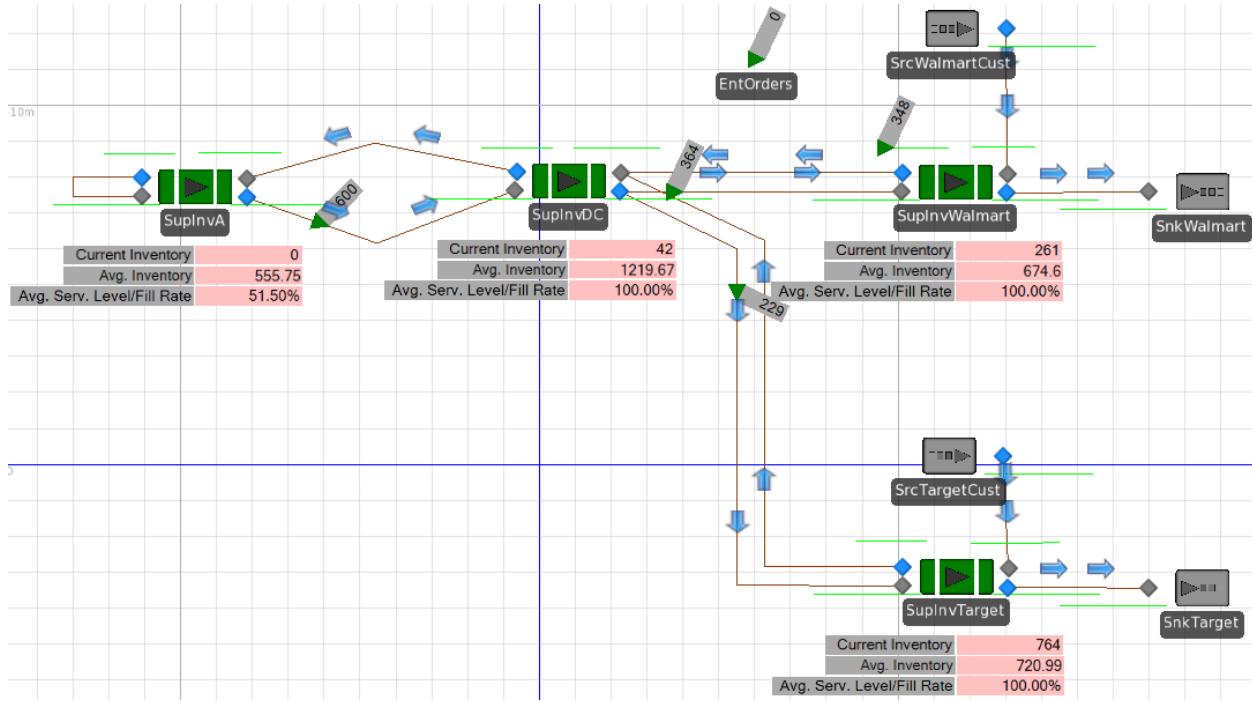
**Step 4:** Figure 22.30.



**Figure 22.30: Setting the Beginning Inventory Position**

## Part 22.5: Using the New to Model the Complex Supply System

The new **SUPPLYINVWORKSTATION** object has been defined and can now be used to build the model (see Figure 22.31) of the complex supply chain system of Figure 22.1.<sup>328</sup>



**Figure 22.31: SIMIO Model of the Complex System**

**Step 1:** In the model “Facility”, add a new **MODELENTITY** named **EntOrders**. Add a status label that is connected to the entity in similar to the one in Chapter 10 with the *Expression* property set to **EStaOrderAmt** as seen in the top right of Figure 22.31.

**Step 2:** Insert two **SOURCES** named **SrcWalmartCust** and **SrcTargetCust** that will represent the daily number of customers arriving at the stores to purchase the products (interarrival times are one each day). As was done before, set the *Before Exiting State Assignments* that will set the **ModelEntity.EStaOrderAmt** to **Random.Poisson(70)** and **Random.Poisson(50)** in days, for the Walmart and Target stores respectively to represent the number of daily orders for each customer type. Here is a case where if we had done the assignment in a tokenized process it would have been easier.

**Step 3:** Next add a new **SUPPLYINVWORKSTATION** named **SupInvWalmart** that will represent the inventory position object of Walmart.

- Set the *Processing Batch Size* to two, *Operation Quantity* to **ModelEntity.StaOrderAmt** and the *Processing Time* property to **Random.Triangular(1,2,3)** in minutes to reflect the processing of the product into the stores and on the shelves as seen in Figure 22.32.

<sup>328</sup> You should spread out the objects until the path direction arrows show. Having paths and connectors in the wrong direction can create some “hard to find” errors. When you are sure the paths and connectors are in the proper direction, place the objects back into their appropriate places.

Properties: SupInvWalmart (SupplyInvWorkStation)	
<input type="checkbox"/> Show Commonly Used Properties Only	
<b>Process Logic</b>	
Capacity Type	Fixed
Ranking Rule	First In First Out
Dynamic Selection Rule	None
+ Transfer-In Time	0.0
Operation Quantity	<b>ModelEntity.EStaOrderAmt</b>
+ Setup Time Type	Specific
+ Setup Time	0.0
Processing Batch Size	2
+ Processing Time	<b>Random.Triangular(1,2,3)</b>
+ Teardown Time	0.0

Figure 22.32: Setting up the Processing of the Batches

- Under the *Inventory* property category, set the *Review Period* to every five days, the *Initial Inventory* to 1100, the *Reorder Point* to 900 with the *Order Up to Qty* to 1100 as seen in Figure 22.33.
- For the *Order Information*, it takes a  $\frac{1}{4}$  of day to process an incoming order. Also, set the *Deliver Node ID* to `Input@SupInvWalmart.ID`<sup>329</sup> which will allow orders sent to the supplier to be sent back to the Walmart store since two different stores are ordering on the same supplier.

<b>Inventory</b>		<b>Order Processing</b>	
InitialInventory	<b>1100</b>	OrderCapacityType	Fixed
ReorderPoint	<b>900</b>	OrderInitialCapacity	Infinity
OrderUptoQty	<b>1100</b>	OrderEntryRankingRule	FirstInFirstOut
+ ReviewPeriod	5	OrderEntryRankingExpression	Entity.Priority
Units	Days	OrderDynamicSelectionRule	None
+ ReviewPeriodTimeOffset	0	+ OrderProcessingTime	<b>0.25</b>
Units	Days	DeliverNodeId	<b>Input@SupInvWalmart.ID</b>
		OrderEntityType	<b>EntReplenishments</b>

Figure 22.33: Inventory and Ordering Information Properties of the Walmart Store

**Step 4:** Repeat Step 3 for the Target store by copying the Walmart **SupInvWalmart** naming the object **SupInvTarget** changing the properties seen in Figure 22.34 making sure to change the *Deliver Node ID* to `Input@SupInvTarget.ID` and the inventory properties since Target does not service as many customers.

<b>Inventory</b>		<b>Order Processing</b>	
InitialInventory	<b>1100</b>	OrderCapacityType	Fixed
ReorderPoint	<b>700</b>	OrderInitialCapacity	Infinity
OrderUptoQty	<b>1100</b>	OrderEntryRankingRule	FirstInFirstOut
+ ReviewPeriod	5	OrderEntryRankingExpression	Entity.Priority
Units	Days	OrderDynamicSelectionRule	None
+ ReviewPeriodTimeOffset	0	+ OrderProcessingTime	<b>0.25</b>
Units	Days	DeliverNodeId	<b>Input@SupInvTarget.ID</b>
		OrderEntityType	<b>EntReplenishments</b>

Figure 22.34: Inventory and Ordering Information Properties of the Target Store

**Step 5:** Connect the two sources to their appropriate **OrderInput** node at the **SupInvWalmart** and **SupInvTarget** via connectors as seen in Figure 22.31.

<sup>329</sup> Recall each object in SIMIO has a unique identification number which is accessed by the `ID` function.

**Step 6:** Insert two **SINKS** into the model named **SnkWalmart** and **SnkTarget**. Connect the **Output** nodes of the **SupInvWalmart** and the **SupInvTarget** **SUPPLYINVWORKSTATIONS** to the respective sinks via CONNECTORS.

**Step 7:** Next we need to create the distribution center which will service the two stores. Copy the **SupInvWalmart** object and name the new **SUPPLYINVWORKSTATION** object **SupInvDC** to represent the DC.

- Set the *Processing Time* property to the following expression in minutes.  
Random.Triangular(5, 10, 20)
- Set the *Initial Inventory* property to 1800, the *Reorder Point* property to 1400 and the *Order Up to Qty* property to 1800.
- Set the *Delivery Node ID* to 0.

**Step 8:** Next we need to connect the DC to each of the two stores.

- For products to flow to the stores, connect the **Output** node of **SupInvDC** as seen in the whole model of Figure 22.31 to the **Input** nodes of **SupInvWalmart** and **SupInvTarget** via **TIMEPATHS**. It takes generally four days to transport product from the DC to each store. However, it can take a minimum of three or a maximum of seven days to travel from the DC to each store. Therefore set the *Travel Time* property to Random.Pert(3, 4, 7) days.
- For orders to flow from the stores to the DC, connect the **ReorderOutput** node of both stores to the **OrderInput** node of the **SupInvDC** via connectors to represent EDI (Electronic Data Information) order transmissions.

**Question 1:** With our new object, how can we model the situation where Walmart orders are satisfied before Target orders, if they arrive at the same time?

---

**Step 9:** In the first portion, we have utilized one supplier to supply the DC, which is a make-to-stock type supplier. Therefore, copy the **SupInvDC** and paste in a new **SUPPLYINVWORKSTATION** object **SupInvA** as seen in Figure 22.35. Only a few things will actually change.

- Set the *Processing Time* property to Random.Triangular(10, 20, 30) (minutes) since it takes longer to convert the raw material into a product.
- Set the *Initial Inventory* property to 600, the *Reorder Point* property to 300 and the *Order Up to Qty* property to 600.
- Set the *Delivery Node ID* to 0.



Figure 22.35: Connecting a Supplier to the DC.

**Step 10:** Similar to the stores, connect the **ReorderOutput** node of the **SupInvDC** to the **OrderInput** node of the **SupInvA** via a connector to have orders flow back to the supplier.

**Step 11:** Like the DC, products take the same amount of time to ship from the Supplier A to the DC. Therefore, connect the **Output** node of **SupInvA** to the **Input** node of the **SupInvDC** via **TIMEPATH** with the same *Travel Time* property set to *Random.Pert(3, 4, 7)* days.

**Step 12:** Supplier A has a perfect raw material supplier (i.e., we assume the material is always available). Therefore, use a connector to connect the **ReorderOutput** of **SupInvA** directly to the **Input** node of **SupInvA** which allows the supplier to order a particular amount of product from its perfect raw material supplier to replenish its inventory position through its own manufacturing.

**Step 13:** Save and run the model for 52 weeks observing the output.

*Question 2:* What are the average service levels of each of the segments in the supply chain?

---

*Question 3:* What is the current and average inventory level of each segment in the chain?

---

*Question 4:* Do you think the current system is adequate and what seems to be the problem?

---

## Part 22.6: Adding a Secondary Supplier for Overflow Orders

By reviewing the results, most of the inventory positions were zero which led to very low service levels. The central issue is the lead time length and variability of getting product from the back of the chain to the front of the chain. There are many ways this situation can be fixed by optimizing reorder points and order up to quantities. Also process improvement projects might reduce the processing times or shipping times. Let's add a second supplier (Supplier B) which is a make-to-order operation to handle orders from the DC as well. In many situations multiple sourcing strategies exist and can be modeled.

**Step 1:** Insert a new **WORKSTATION** named **WrkSupB** to act as a make-to-order supplier (i.e., when orders arrive they are produced and not stored ahead of time). Modify the model as shown in Figure 22.36.

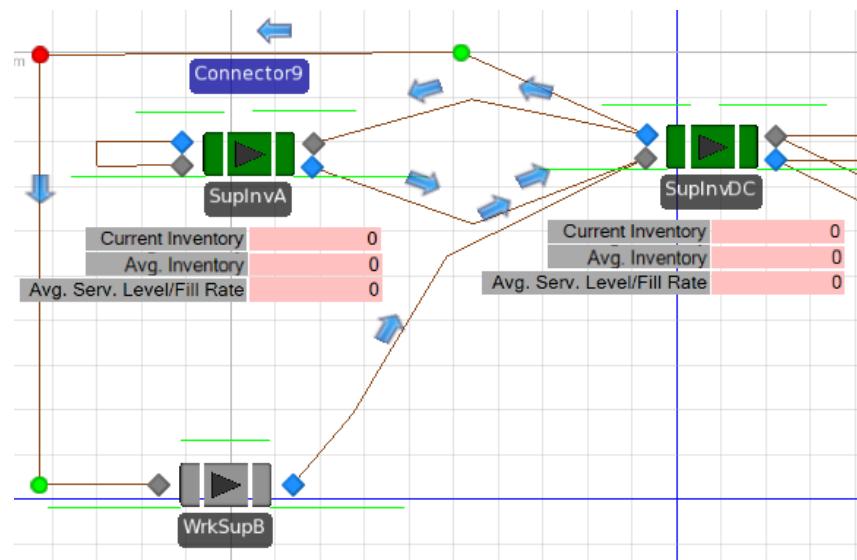
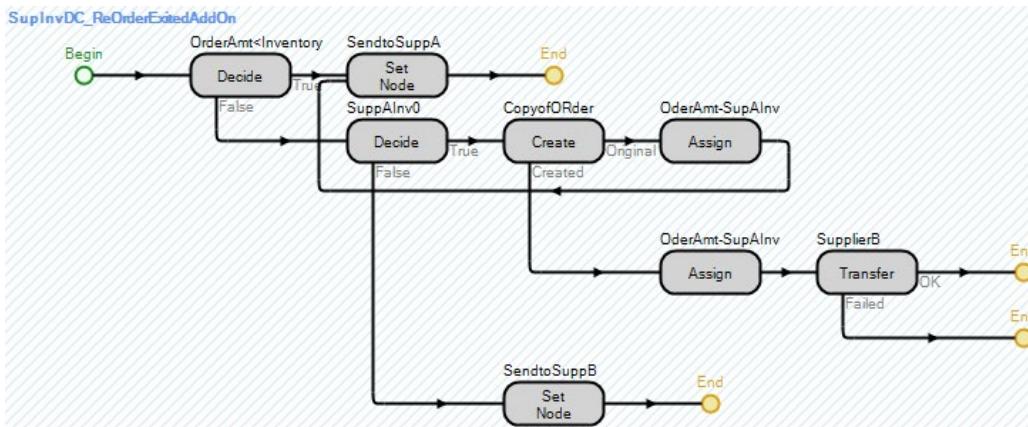


Figure 22.36: Add Make-to-Order Supplier B

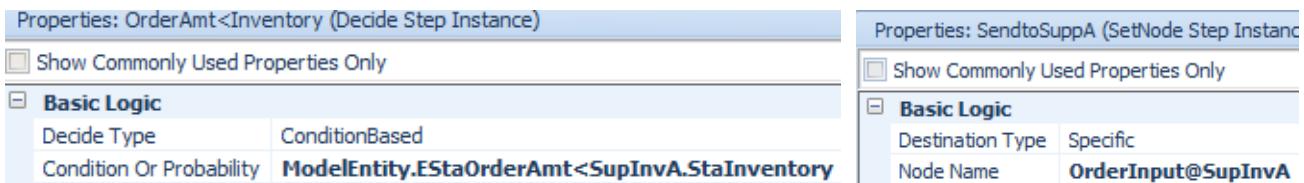
- Connect the **ReorderOutput** of **SupInvDC** to the **Input** of the **WrkSupB** via a **CONNECTOR** link to allow orders to be sent to the new supplier electronically.
- To allow products to flow, connect the **Output** node of the **WrkSupB** to the **Input** node of the **SupInvDC** via a **TIMEPATH**. It takes **Random.Pert(1,2,3)** days to ship then product to the DC from this supplier.
- Set the *Processing Time* property of the **WrkSupB** to produce the products to **Random.Triangular(8,10,20)** (minutes). Again set the *Operation Quantity* to **ModelEntity.EStaOrderAmt** and the *Processing Batch Size* property to two.

**Step 2:** In this example, the DC has access to the Suppliers A current inventory level. If the order amount is greater than the current inventory level, the DC will order only up to the inventory position and then order the remaining portion from Supplier B. Insert a new “*ReOrderExited*” add-on process trigger that is executed when the order exits the **WrkSupB** as seen in Figure 22.37.



**Figure 22.37: ReOrder Exit Add-on Process Trigger**

- Insert a *Decide* step that checks to see if the **ModelEntity.EStaOrderAmt** is less than or equal to the **SupInvA.Inventory** position. If it is (i.e., the “True” branch) then use a *Set Node* step to specify that the order should be sent to the Supplier A (i.e., **OrderInput@SupInvA**).



**Figure 22.38: The Properties of the *Decide* and *Set Node* for Reordering Process**

- If the order amount is bigger than the current inventory position then decide if supplier A has any inventory at all (i.e., **SupInvA.StaInventory >0**). If it does not (i.e., “False” branch) then send the current order to supplier B via the *Set Node* with *Node Name* property set to **Input@WrkSupB**. Otherwise, *Create* a copy of the current order and set the “Original” order amount to the current inventory level as seen in Figure 22.39.

Properties: SuppAInv0 (Decide Step Instance)		Properties: CopyofORder (Create Step Instance)	
<input type="checkbox"/> Show Commonly Used Properties Only		<input type="checkbox"/> Show Commonly Used Properties Only	
<input checked="" type="checkbox"/> Basic Logic		<input checked="" type="checkbox"/> Basic Logic	
Decide Type	ConditionBased	Create Type	CopyAssociatedObject
Condition Or Probability	SupInvA.StaInventory>0	Entity Type	
		Number Of Objects	1

Figure 22.39: Create Step to Copy the Current Order

- In the assignment, as shown in Figure 22.40 set the inventory position at **SupInvA** to the **StaOrderAmt** and the original order's **StaOriginalOrderAmt** needs to be set equal to the new **StaOrderAmt**. Then go ahead and send the order on to the **SupInvA** by moving the "End" point on top of the *Set Node* as seen in Figure 22.37.

<input checked="" type="checkbox"/> Basic Logic		<input checked="" type="checkbox"/> Basic Logic	
State Variable Name	ModelEntity.EStaOrderAmt	State Variable Name	ModelEntity.EStaOriginalOrderAmt
New Value	SupInvA.StaInventory	New Value	ModelEntity.EStaOrderAmt

Figure 22.40: Set Order Amount and the Original Order Amount

- For the newly created order which is a copy, set the **StaOrderAmt** to the remaining portion and also update the **StaOriginalOrderAmt** as shown in Figure 22.41.

<input checked="" type="checkbox"/> Basic Logic		<input checked="" type="checkbox"/> Basic Logic	
State Variable Name	ModelEntity.EStaOrderAmt	State Variable Name	ModelEntity.EStaOriginalOrderAmt
New Value	ModelEntity.EStaOrderAmt-SupInvA.StaInventory	New Value	ModelEntity.EStaOrderAmt

Figure 22.41: Creating an Order to Send to the Second Supplier

**Step 3:** Then transfer from "Free Space" to **Input@WrkSupB** as seen in Figure 22.42.

Properties: SupplierB (Transfer Step Instance)	
<input type="checkbox"/> Show Commonly Used Properties Only	
<input checked="" type="checkbox"/> Basic Logic	
From	FreeSpace
To	Node
Node Name	Input@WrkSupB

Figure 22.42: Transfer Input at WrkSupB

**Step 4:** Save and run the model for 52 weeks observing the output.

**Question 5:** What are the average service levels of each of the segments in the supply chain?

**Question 6:** What is the current and average inventory level of each segment in the chain?

## Part 22.7: Commentary

- This model provides a framework for many interesting embellishments and much added realism.
- Failure information could be linked with the **ResOrder** resource. The ability to suspend the ordering process when failures occur should also be added.
- Cost information can easily be added to the orders and objects using the cost parameters under the financials and one can choose suppliers as well as have different shipping options (i.e., air, rail, and trucking) to assist in meeting consumer demand to make it even more realistic.

- The reliability of the supplier can be added to the new **SUPPLYINVWORKSTATION** object such that shipping or picking errors can be modeled (suppose a customer asks for 100 but only 98 are shipped, even though there is sufficient inventory).
- The object can be modified to handle multiple products rather than a single product type.
- Additional inventory policies can be built into the model that would allow the user to choose the particular inventory policy for each segment of the supply chain.
- We could also add a **MONITOR** event to create a continuous review system rather than just a periodic review system. Users could choose between the systems.
- The **WORKSTATION** was chosen as the object to subclass rather than the **SERVER** since it can process a batch of parts correctly. However, you cannot increase the capacity of **WORKSTATIONS** like you can **SERVERS** by changing the capacity. You can change the batch size but that is not quite the same as the average flow time for the parts would be correct but the variance would not. One would have to insert multiple workstations to achieve the same thing.

# Chapter 23

## More Subclassing: Process Planning/Project Management

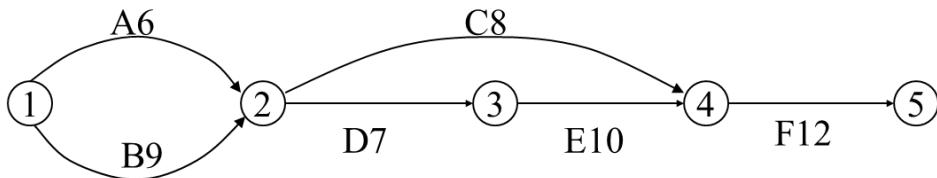
### Part 23.1: Process Planning

A new product (called Product 3) is being introduced. One unit of Product 3 is produced by assembling one unit of Product 1 and one unit of Product 2. Before production begins on either Product 1 or Product 2, raw materials must be purchased, and workers must be trained. Before Products 1 and 2 can be assembled into Product 3, the finished Product 2 must be inspected. The table below gives the list of activities and their predecessors and the average duration of each activity.

**Table 23.1: Activity Precedence and Duration**

Activity	Predecessors	Duration (days)
A = train workers	--	6
B = purchase raw materials	--	9
C = produce Product 1	A, B	8
D = produce Product 2	A, B	7
E = test Product 2	D	10
F = assemble Products 1 and 2	C, E	12

The typical project management diagram for example can be seen in Figure 23.1.



**Figure 23.1: Sample Project Planning Example**

Clearly, the project will take an average of 38 days, and activities B, D, E, and F are critical (“on the critical path”), meaning if any of these increase, then the time to completion will increase. However, in reality, activity completion times are uncertain (random), and different activities may become critical. So, we can’t rely on deterministic values for the activity times; instead, we need to represent the times with random variables.

Also, activities might share resources to complete the task. Therefore, the goal will be to determine the critical path and the probability that a path will be on the critical path.

**Table 23.2: Project Activity Statistics**

Earliest Finish	The time the current task finishes (i.e., entity leaves the path).
Latest Finish	The time the current task could finish without changing the critical path or causing the successive tasks to start later.
Slack	The time difference between the Earliest Finish and the Latest Finish
Percent CP	The percentage of the time the current path is on the critical path.

Earlier chapters have demonstrated the ability to specialize/subclass existing objects, which allows additional characteristics and behaviors to be added or existing ones to be modified, but none have specialized one of the link objects (i.e., PATHS or TIMEPATHS). We will model the project management problem using the TIMEPATHS

to consume time. We will also use one replication to represent a single project completion, so multiple replications will be used to characterize the project statistically.

## Part 23.2: Creating a Specialized TIMEPATH to Handle Activities

The ability to create new objects is one of the advantages of SIMIO, and the TIMEPATH link is an attractive object to use for the activity time since the entities can be seen visually moving along the paths. The paths represent the important objects (i.e., critical path determination). Each path needs to keep track of its own statistics (i.e., Earliest Finish, Slack, Percent CP, etc.). The path also needs to know when an entity reaches the end of the path and restrict only one entity from traveling on each path. This encapsulation of the time path gives the object the “intelligence” to calculate the statistics automatically and allows us to utilize it for many different project management problems.

**Step 1:** Create a new model, subclass the TIMEPATH, and change the *Model Name* property to **PRJTIMEPATH**<sup>330</sup>.

**Step 2:** Select the **PRJTIMEPATH** from the [Navigation] panel. Insert two DISCRETE REAL STATE variables named **StaEarliestFinish** and **StaCalculateSlack**. The first state variable will be used to keep track of the time the activity finishes, while the second one will be used to signal when to calculate the slack.

**Step 3:** To keep track of critical path calculations, add three TALLY statistics named **TallyStatEarliestFinish**, **TallyStatSlack**, and **TallyStatPercentCP** via the *Definitions*→*Elements* section.

- Change the “Category” property to “Critical Path” for all three TALLY statistics under the “Results Classification.”
- Change the *Data Item* property to “Earliest Finish,” “Slack,” and “Percent CP” for the appropriate statistic.
- Also, for the **TallyStatEarliestFinish** and **TallyStatSlack**, set the *Unit Type* to “Time.”

**Step 4:** Recall only one project will be completed for each replication. In order to model a “junction” from which multiple activities start, the paths should not allow more than one traveler, and travel should be in one direction only. From the *Definitions*→*Properties*, click on the down arrow to reveal the inherited properties.

- Select the *InitialTravelerCapacity* property and change the *Default Value* from Infinity to **one** and the *Visible* property to “**False**” so a user cannot change it, and it will automatically be set.
- Select the *InitialDesiredDirection* property and change the *Default Value* to **forward** and the *Visible* property to “**False**” so a user cannot change it, and it will automatically be set.

---

<sup>330</sup> Right click on the object in the [Navigation] section and select properties.

Properties: InitialTravelerCapacity (Expression Property)	
Value	
Default Value	1
Switch Property Name	
Switch Condition	Equal
Switch Value	
Candidate References	False
Unit Type	Unspecified
Appearance	
Display Name	Initial Traveler Capacity
Category Name	Travel Logic
Expanded	False
Parent Property Name	
General	
Name	InitialTravelerCapacity
Description	All instances of this property
Required Value	True
Visible	False

Properties: InitialDesiredDirection (Enumeration Property)	
Value	
Default Value	Forward
Switch Property Name	Type
Switch Condition	Equal
Switch Value	Bidirectional
Enum Type	TrafficDirection
Appearance	
Display Name	Initial Desired Direction
Category Name	General
Expanded	False
Parent Property Name	Type
Captions	
General	
Name	InitialDesiredDirection
Description	All instances of this property
Required Value	True
Visible	False

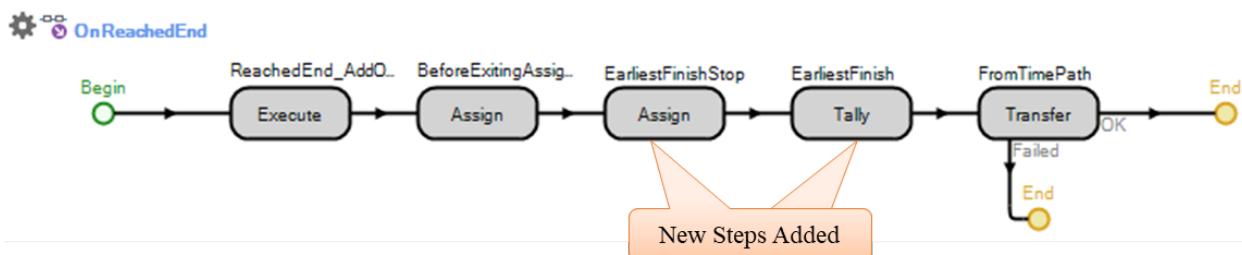
**Figure 23.2: Setting the Default Values of the Travel Capacity and Desired Direction**

**Step 5:** The TIMEPATH defines seven processes that determine the behavior of the link, as seen in Table 23.3.

**Table 23.3: Processes of the TimePath**

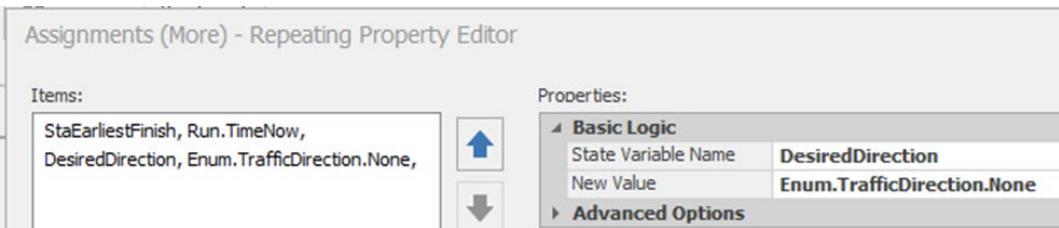
Process	Description
<i>OnEntered</i>	Process runs when the leading edge of an Entity, Worker, or Transporter reaches the beginning of the TIMEPATH.
<i>OnExited</i>	Process runs when the trailing edge of an Entity, Worker, or Transporter leaves the TIMEPATH.
<i>OnReachedEnd</i>	Process runs when the leading edge of an Entity, Worker, or Transporter reaches the end of the TIMEPATH.
<i>OnRunEnding</i>	Process runs once at the end of the simulation.
<i>OnRunInitialized</i>	Process runs once at the beginning of the simulation.
<i>OnTrailingEdgeEntered</i>	Process runs when the trailing edge of an Entity, Worker, or Transporter enters the path.

**Step 6:** Once the project ENTITY reaches the end of the path, the activity is finished, and the earliest finish time can be calculated. Override the “*OnReachedEnd*” process as seen in Figure 23.3, to perform the calculation. 6.5



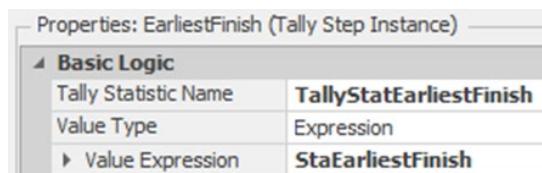
**Figure 23.3: Calculating Earliest Finish and Shutting Down Path**

- Insert an Assign step that will set the DesiredDirection to Enum.TrafficDirection.None will shut down the path and not allow any more entities to travel this path. Also, set the StaEarliestFinish state variable to the current time (Run.TimeNow). Refer to Figure 23.4.



**Figure 23.4: Shut down the Path and Set the Earliest Finish**

- Insert a *Tally* step to add one observation of the **TallyEarliestFinish** statistic, as seen in Figure 23.5.



**Figure 23.5: Add an Observation of the Earliest Finish Statistic**

### Part 23.3: Creating a Junction Object to Handle Precedent Constraints

In the previous section, we modeled the project activities as specialized **TIMEPATHS**, automatically calculating all the statistics. However, they cannot enforce the precedence constraints or start-up activities nor determine when the latest finish happens. The precedence constraints have to be enforced by the junctions, which also initiate the next set of activities. Therefore, the junction has to wait until all the preceding activities have finished before starting all of the succeeding activities at the same time. Also, the junctions determine the latest finish of all the preceding activities based on the time the last of the preceding activities finishes.

**Step 1:** None of the standard objects can model this circumstance easily, and most standard objects are too complicated to modify for this purpose. A new object similar to the **DELAY** object of Chapter 20 will be created. From the “*Project Home*” tab, insert a new “Fixed Class Model” named **JUNCTION**, which will be used to model the junctions.

**Step 2:** We need our junction to act as a start node and an end node, as well as intermediate junctions within the project network. Start nodes will start activities at the start of the project, while the end nodes will terminate all activities at the end of the project.

- Insert a new String **LIST** from the *Definitions*→*List*→*Create* section named **ListJunctionType** with three entries, as seen below. This will allow the user to choose the Junction Type.

▲ Strings	
ListJunctionType	
	String
► 0	<b>Junction</b>
1	<b>StartNode</b>
2	<b>EndNode</b>

**Figure 23.6: Defining Different List Types**

- Insert a new standard “List Property” named **JunctionType**, which allows the modeler to specify the junction type with “Junction” as the default type and place it in the “Project Management” category, as seen in Figure 23.7.<sup>331</sup>

Properties: JunctionType (List Property)		
Value		
Default Value	Junction	
Switch Property Name		
Switch Condition	Equal	
Switch Value		
List Name	ListJunctionType	
Appearance		
Display Name	JunctionType	
Category Name	Project Management	
Expanded	False	
Parent Property Name		
Captions		
Junction	<input checked="" type="checkbox"/>	Junction
StartNode	<input checked="" type="checkbox"/>	Start Node
EndNode	<input checked="" type="checkbox"/>	End Node
General		
Name	JunctionType	
Description	Specify whether this junction is a start, end, or junction type	
Required Value	True	
Visible	True	

Figure 23.7: Specifying the List JunctionType Property

**Step 3:** If the **JUNCTION** object is a “Start Node,” it will need to create entities to be sent out into the project network like the standard **SOURCE**. From the *Definitions→Properties→Add→Object Reference*, add an **Entity** property named **EntityType**, making sure to place it into the “Project Management” category created in the step before. Also, you only want the property displayed if the **JunctionType** property is set to “StartNode.”

Properties: EntityType (Object Property)		
Value		
Default Value	null	
Switch Property Name	JunctionType	
Switch Condition	Equal	
Switch Value	StartNode	
Filter to Resources	False	
Object Type		
Default Value Instantiation	None	
Appearance		
Display Name	EntityType	
Category Name	Project Management	
Expanded	False	
Parent Property Name		
General		
Name	EntityType	
Description		
Required Value	True	
Visible	True	

Figure 23.8: Specifying the Start Node

<sup>331</sup> Note, that you can change the caption of the list object as a space was added between “*StartNode*” and “*EndNode*” options.

**Step 4:** Each junction needs to know the number of precedent activities to wait on and the number of successive activities to start. Insert two new DISCRETE INTEGER STATE properties named **StaPredecessors** and **StaSuccessors**, which will be set at the beginning of the simulation run.<sup>332</sup>

**Step 5:** Insert a DISCRETE INTEGER STATE variable named **StaActivitiesCompleted**, which will keep track of the number of activities that have been completed, and a DISCRETE REAL STATE variable named **StaLatestFinish**, which will record when the time the last activity has been completed.

**Step 6:** As was done in Chapter 20, the **DELAY** object, the incoming entities need a location where actions can occur. From the *Definitions→Elements*, insert a new STATION named **Project** with all the default values.

**Step 7:** Next, let's define the external view seen by the user of the **JUNCTION** object.

- From the *Definitions→External→Drawing* section, use a Polygon shape to draw a red diamond, as seen in Figure 23.9.
- Insert an EXTERNAL NODE that is a BASICNODE on the left edge of the red diamond that will accept incoming activities and automatically send them to the **Project** STATION.
- Next create the output of activities by inserting an External Node that is a TRANSFERNODE on the right edge of the red diamond that will route successor activities.

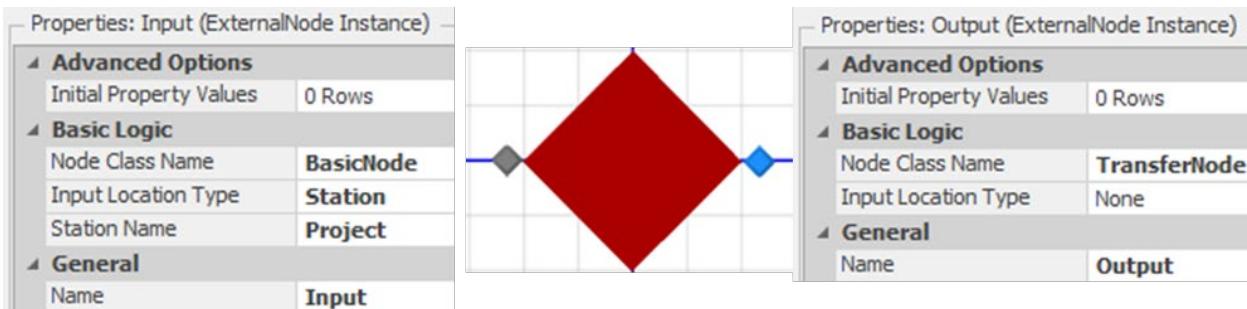


Figure 23.9: External View of the Junction Object Along with Input and Output Nodes

Now that we have defined the external view and activities (i.e., entities) that can flow in and out of our object, we need to add the process logic to handle the precedence and the starting of the activities as well as determining the number of predecessors and successors.

**Step 8:** From the “Process” tab, select the **OnRunInitialized** process from the *Select Process* dropdown. Insert an *Assign* step that will set the number of predecessors (i.e., **StaPredecessors** to **Input.InboundLinks.NumberItems**) and the number of successors (i.e., **StaSuccessors** to **Output.OutboundLinks.NumberItems**).

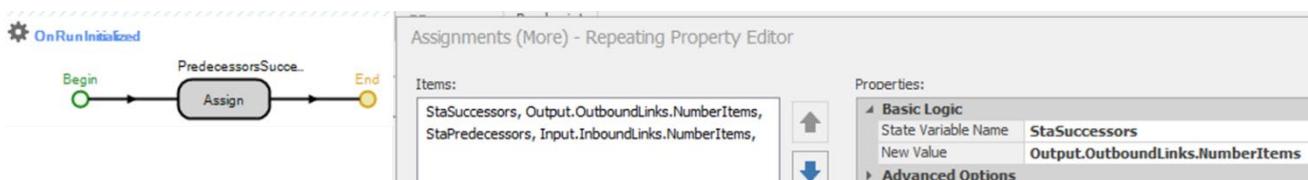


Figure 23.10: Initializing the Number of Successors and Predecessors

<sup>332</sup> In Versions prior to 5 there was no way to access the number of links coming or going out of BASICNODES or TRANSFERNODES and these values had to be passed in as a number.

**Step 9:** Create a new process named “*OnEnteringProject*” that will execute every time an activity (i.e., ENTITY) enters the **Project** station by specifying the *Triggering Event* to be “*Project.Entered*,” as seen in Figure 23.11. The process needs to update the number of activities finished. If the number of activities equals the number of predecessors, it will create the number of successors and send them out to the network. Otherwise, activities still need to be completed, and we will destroy the current activity entity.

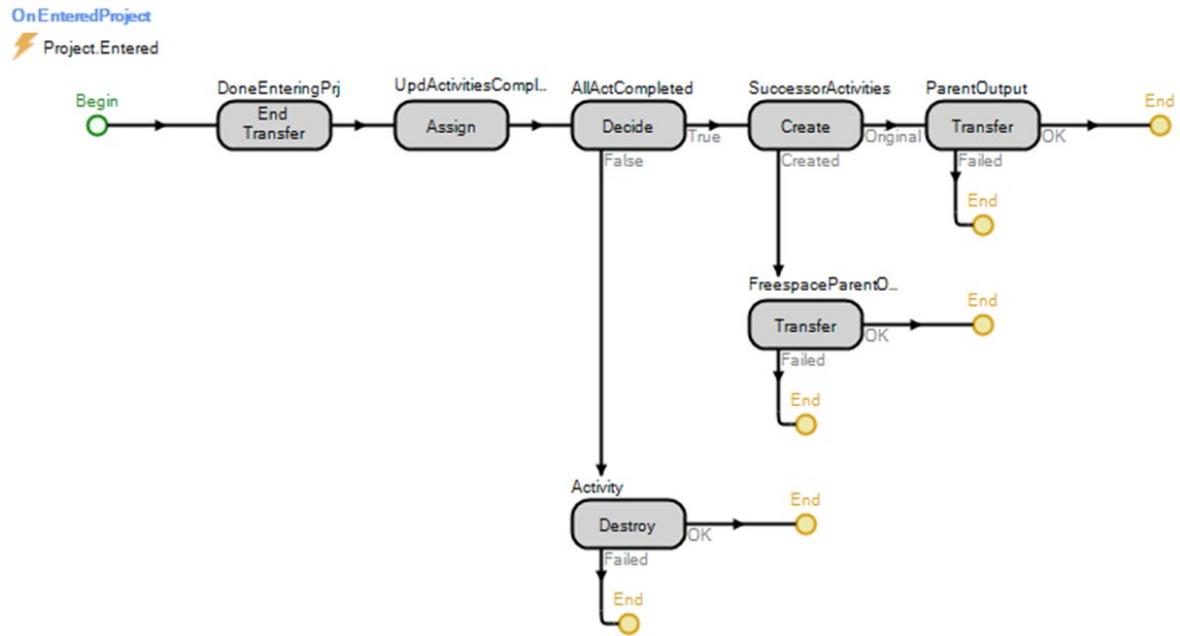


Figure 23.11: Process to Handle Activities that Enter the Junction

- As always, you first need to end the transfer of the entity object into the station by inserting an *End Transfer* step.
- Next, update the number of activities that have been completed by one.

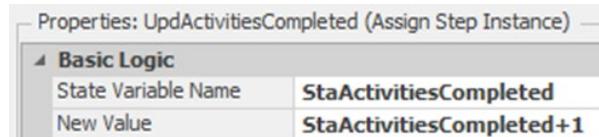


Figure 23.12: Updating the Number of Activities by One

- Now, decide if all the precedent activities have been completed (i.e., *StaActivitiesCompleted == StaPredecessors*) as seen in Figure 23.13.

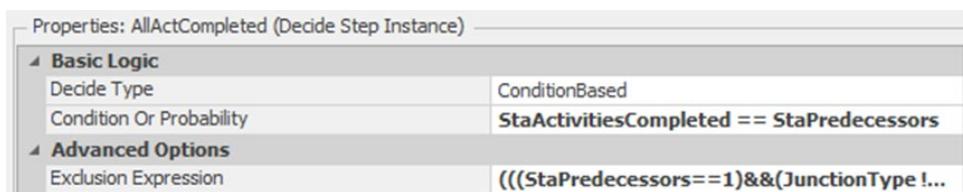
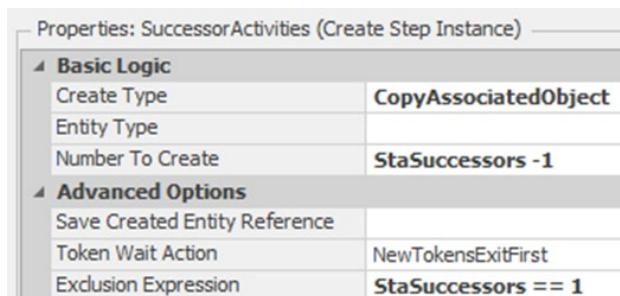


Figure 23.13: Deciding if All Preceding Activities are Complete

- Note the use of the *Exclusion Expression* property, which can be used at the start of the simulation to determine if this step is necessary.<sup>333</sup> If the **JUNCTION** is a “*StartNode*” or only has one preceding activity, then the next set of activities can be automatically started (it will be set to one), or if this is an “*EndNode*,” then these entities can be automatically destroyed (i.e., set to two). If you know at the start of the simulation, skipping steps can greatly speed up the simulation since it does not need to be evaluated. Set the Exclusion property to the following.

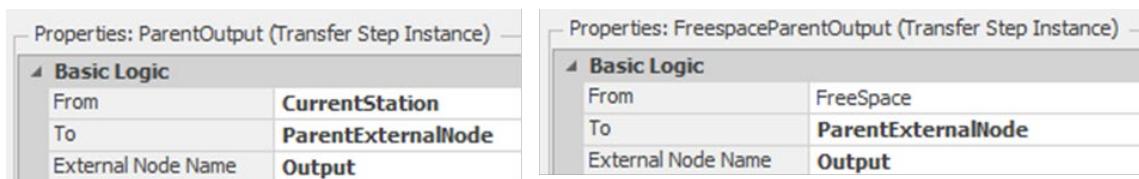
```
(( (StaPredecessors==1) && (JunctionType != List.ListJunctionType.EndNode) )
|| (JunctionType == List.ListJunctionType.StartNode) + 2*(JunctionType
== List.ListJunctionType.EndNode)
```

- If the total number of preceding activities has not been completed (i.e., some activities are still waiting to be completed), then insert a *Destroy* step to kill the current activity entity.
- If this activity is the last one to complete or the junction is a “*Start Node*,” then we need to start the next set of successor activities. Insert a *Create* step that will create copies of the current Entity (i.e., set the *Create Type* property to “*CopyAssociatedObject*”). The *Number of Objects* will be set to *StaSuccessors -1* since one activity **ENTITY** is already present. Note the use of the *Exclusion Expression* property to skip this step if there is only one successor, as seen in Figure 23.14.



**Figure 23.14: Creating the Successor Activities**

- Using *Transfer* steps, the original activity **ENTITY** needs to be sent from the “*Current Station*” to the “**ParentExternalNode**” **Output**, while the newly created activity entities need to be sent from “*Free Space*” to the same **ParentExternalNode Output** as seen in Figure 23.15.

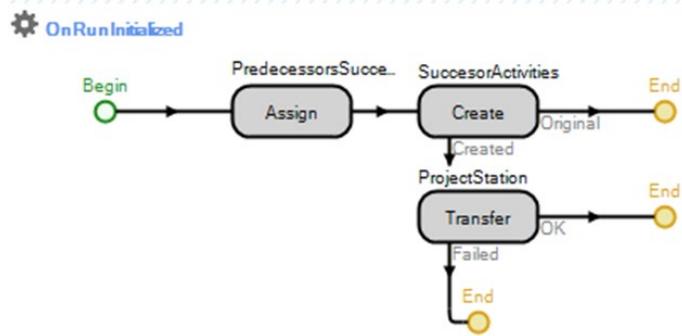


**Figure 23.15: Sending Successor Activities Out to the Network**

**Step 10:** If the **JUNCTION** is a “*Start Node*,” then it must automatically create the start activity to start this branch of the network. Therefore, the “*OnRunInitialized*” process which is executed for each object at the beginning of the simulation, will be used to handle this logic. Select the “*OnRunInitialized*” and add a *Create* and *Transfer* steps after the *Assign* sets the number of predecessors and successors.

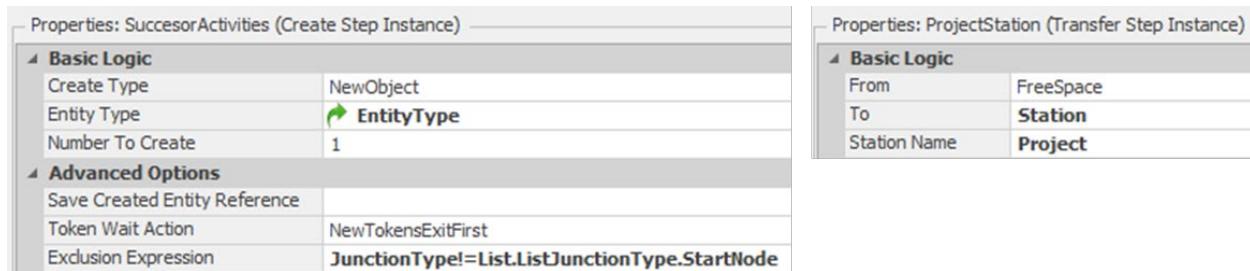
---

<sup>333</sup> If the expression evaluates to a one then the step will not be included and the entity will flow directly out of the true or primary branch. If it evaluates to a two then it would be skipped and flow directly out of the false or secondary branch. All other evaluations will have the process step evaluated each time.



**Figure 23.16: “OnRunInitialized” Process to Start Activities**

- Insert a *Create* step that will produce one new object of *EntityType* property. Again, right-click on the *Object Instance Name* property and select the referenced property. Again, note the use of the *Exclusion Expression* property to skip this step if it is not a “Start Node.” Refer to Figure 23.17.
- If it is a start node and an activity has been created, then insert a *Transfer* step to move it from “FreeSpace” to the **Project Station**.<sup>334</sup> This will then run the “*OnEnteringProject*” process, which will produce the correct number of successors without repeating the code, as seen in Figure 23.17.



**Figure 23.17: *Create* and *Transfer* Steps for the “OnRunInitialized” Process**

#### Part 23.4: Creating Small Network to Test the New Object

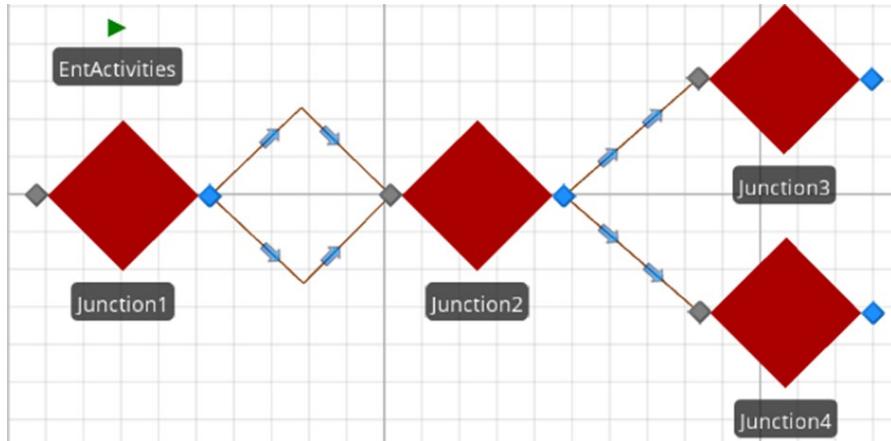
At this point, our junction object seems to be set up to handle the precedence constraints and start new activities. Go back to the original model and create a very simple network to test this ability. At this point, we will not have to worry about naming the objects.

**Step 1:** Insert a new MODEL ENTITY named **EntActivities** and four Junction objects, leaving their names as the default. Place them in the configuration as seen in Figure 23.18.

String.format("The successor or predecessor {0} connected to {1} is not a PrjTimePaths. Change the link to PrjTimePath",Link.Name, Name)

---

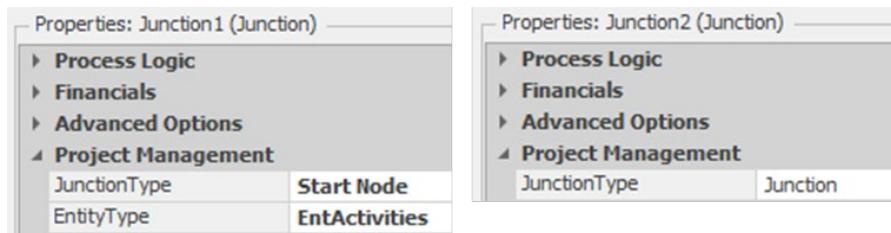
<sup>334</sup> Note you could have sent the entity from “FreeSpace” to the **Input ParentExternalNode** as well. Either way it runs the process.



**Figure 23.18: Simple Network to Test the JUNCTION Object**

**Step 1:** Connect **Junction1** to **Junction2** via two **PRJTIMEPATHS** with one path taking one day and the other one taking two days and then connect **Junction2** to **Junction3** and **Junction4** via one day **PRJTIMEPATHS**.

**Step 2:** Set **Junction1** to be a “Start Node” while **Junction2** should be just a “Junction,” as seen in Figure 23.19. Set **Junction3** and **Junction4** to each be an “End Node.” Set the model to run for four days.



**Figure 23.19: Setting up the Start Junction and the Secondary Junction**

**Step 3:** Save and run the model, observing what happens.

**Question 1:** Did two activities get created from Junction1 and Junction2?

---

**Question 2:** Did Junction2 wait before both activities were completed before starting the next set of activities?

---

**Step 4:** It looks like only one was created when actually both were placed on the same path. Click on the **Output** node of **Junction2** and change the *Outbound Link Preference* from “Any” to “Available,” which is used to select the next link. In this case, we only want to select from the currently available ones.

**Step 5:** Save and run the model, observing what happens.

**Question 3:** Did two activities get started from Junction2 now?

---

**Step 6:** At this point, the user will have to change the default property from “Any” to “Available” every time which is not acceptable since they may forget. Therefore, we will automatically set it for the modeler by specializing the **TRANSFERNODE** object. Subclass the **TRANSFERNODE** object and name it **PrjTransferNode**.

**Step 7:** Select the **PRJTRANSFERNODE** from the [Navigation Panel]. From the *Definitions→Properties*, click on the “Properties (Inherited)” down arrow to show all the inherited properties. Select the **OutboundLinkPreference** property, change the *Default Value* to “Available,” and set the *Visible* property to “False” so the user can’t accidentally set it back.

**Step 8:** Return back to the **JUNCTION** object and select the external **Output** node in the *Definitions→External* section. Change the *Node Class* property from the **TRANSFEROBJECT** to the new **PRJTRANSFERNODE**, as seen in Figure 23.20.

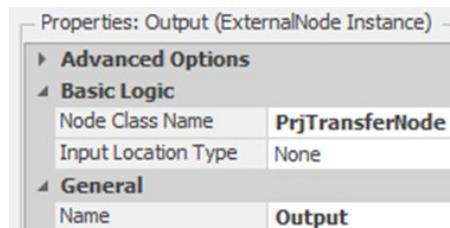


Figure 23.20: Changing the Output node to be PrjTransferNode

### Part 23.5: Building the Example Network

Let’s build the simple example from the beginning section.

**Step 1:** From *Project Home*, create a new *Fixed Model* and insert **JUNCTIONS** and a **MODELENTITY** using the names as shown in Figure 23.21.

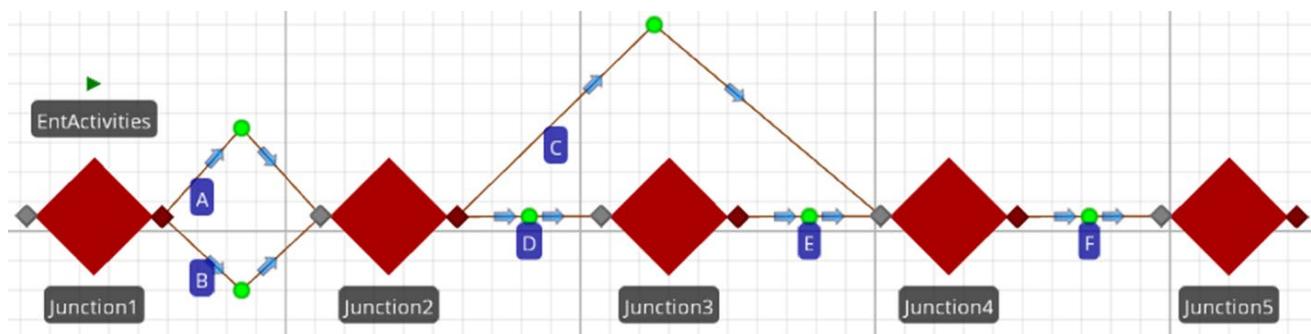


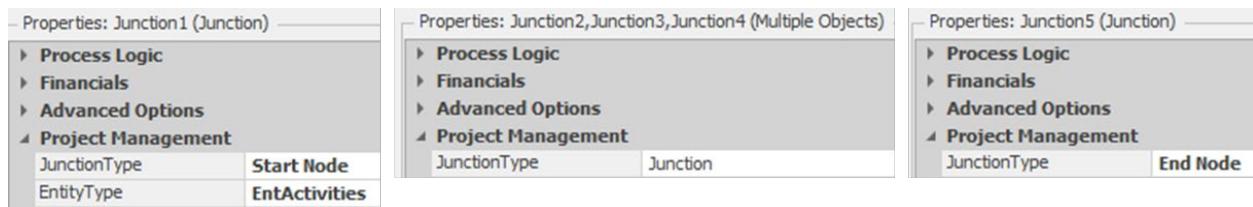
Figure 23.21: Example Project Management Model

**Step 2:** Use **PRJTIMEPATHS** to represent each of the activities by connecting two **JUNCTIONS** together. Table 23.4 should be utilized to specify the name of each path and set the *Travel Time* and *Units* properties.

Table 23.4: Activity Times and Names

From	To	Name	Travel Time	Units
Junction 1	Junction 2	A	Random.Pert(5, 6, 10)	Days
Junction 1	Junction 2	B	Random.Pert(6, 9, 9)	Days
Junction 3	Junction 4	C	Random.Pert(6, 8, 16)	Days
Junction 2	Junction 3	D	Random.Pert(4, 5, 7)	Days
Junction 2	Junction 4	E	Random.Pert(6, 7, 10)	Days
Junction 4	Junction 5	F	Random.Pert(8, 12, 16)	Days

**Step 3:** Next, use the values in Figure 23.22 to specify each of the five junctions’ Project Management properties to ensure the precedent constraints are satisfied and the number of successor activities are started.



**Figure 23.22: Properties of the Five Junctions**

**Step 4:** Save and run the model, and make sure the run length is “Unspecified,” which will stop the simulation when no more entities are in the system.

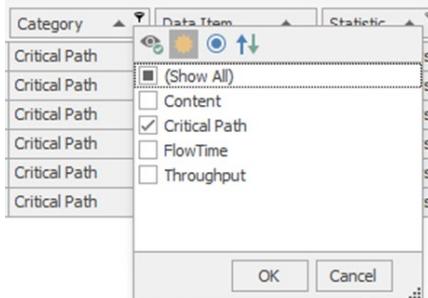
**Question 4:** How many projects were completed, and what was the total completion time?

---

**Step 5:** Remember, we decided to model one project completion with one replication. Therefore, insert a new experiment that will replicate the project management network 1000 times.

**Step 6:** Run the experiment and look at the results under the “Pivot Grid” tab. Change the unit settings from time to days.

**Step 7:** Filter the “Category” column by clicking on the funnel in the right-hand corner (see Figure 23.23) and, select only the “Critical Path” statistics, and then filter the “Statistic” column to include the “Average” since we only have one observation per replication. Figure 23.24 shows the reduced statistics where the project will take between 27 and 38 days, with the average taking 33 days plus or minus 1.8 days<sup>335</sup>.



**Figure 23.23: Filter the Category only to Include Critical Path Statistics**

Scenario 1					
Object Type	Object Name	Data Source	Category	Data Item	Statistic
PrjTimePath	A	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)
	B	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)
	C	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)
	D	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)
	E	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)
	F	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)

**Figure 23.24: Earliest Finish Statistics for 1000 Replications**

---

<sup>335</sup> Your numbers may be a little different since the random numbers may be sampled differently in your model, due to the way the modeling components were added. However, the results should be statistically identical.

## Part 23.6: Adding the Slack and Percent of Time on Critical Path Calculations

At this point, we can determine the Earliest Finish (EF) time of all the paths and the total time to complete the project, but it is difficult to determine what paths constitute the critical path since path times are random. However, the slack time for each path and the percentage of time this path is on the critical path can assist the project manager. The Earliest Finish (EF) time can be calculated without any other information from other objects. However, the computation of the Latest Finish (LF) requires knowing the slack and the number of times the path is on the critical path.

The Latest Finish time is only known at the junctions and is determined when the last activity finishes. When two SIMIO objects need to communicate with one another, there are two ways to approach this communication. In the first approach shown in Figure 23.25, the **JUNCTIONS** need to know the **PRJTIMEPATHS** that are connected to them. The **PRJTIMEPATHS** will monitor a state variable within their object that is set to one by the **JUNCTION** when the last activity finishes. In the second approach, the **PRJTIMEPATHS** need to know the **JUNCTIONS** to which they are connected, and they will monitor a state variable inside the **JUNCTION** object. We will illustrate the first approach in this section, where the monitoring state variable resides inside the object that is monitoring.

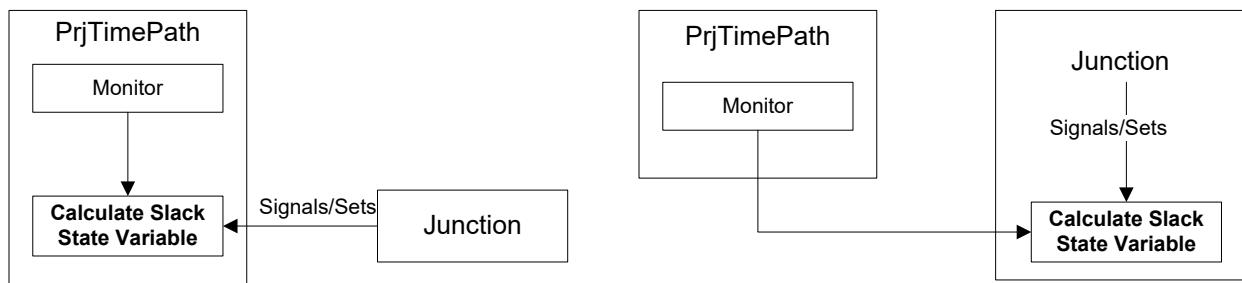


Figure 23.25: Approaches to Signaling from One Object to Another

**Step 1:** Save the current model as a **Chapter 23-6.spfx** because we will need to return to the previous section model to create the second approach, so we need to ensure it is saved before renaming it.

**Step 2:** Select the **PRJTIMEPATH** object from the [Navigation] Panel.

**Step 3:** From the *Definitions→Elements* tab, insert a new **MONITOR** named **MonitorSlack**, which will be used to calculate the slack calculation when the **StaCalculateSlack** state variable changes values to one, as seen in Figure 23.26.

- Set the *Monitor Type* to “DiscreteStateChange” and the *State Variable Name* to the **StaCalculateSlack** variable.
- Create a new *Triggered Process Name* to execute when the monitor event changes.

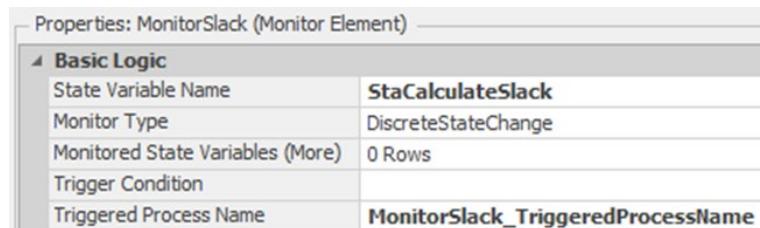


Figure 23.26: Setting up the Monitor to Calculate the Slack

**Step 4:** Once the last activity has happened, the associated **JUNCTION** will set the **StaCalculateSlack** state variable to one. Since this variable is being watched by the **MonitorSlack**, it will execute the “*Triggered Process*,” as seen in Figure 23.27.

- Insert a *Tally* step that will calculate the slack (LF-EF) where the latest finish is the current time and add either a zero or one to the **TallyPercentCP** to indicate whether or not the path is on the critical path or not. You need to select the *Tallies (More)* repeating group to set both statistics.

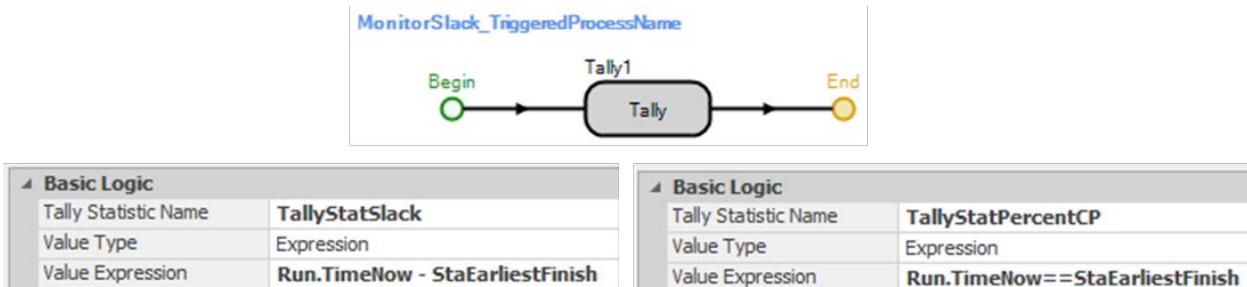


Figure 23.27: Calculating the Slack and Determining if it is on Critical Path

**Step 5:** The **PRJTIMEPATH** object is now setup to calculate the two new statistics when its **StaCalculateSlack** variable is changed. The **JUNCTION** object must now set the **StaCalculateSlack** variable for all precedent paths once the last activity has finished. Select the **JUNCTION** object from the [Navigation] panel.

**Step 6:** After the last precedent activity has finished (**JUNCTION** object's *OnEnteringProject* process), the **StaCalculateSlack** variable will need to be set to one for all the precedent paths. Insert a *Search* step that will be used to search for the correct activity paths right after the *Decide* step. Recall the search step, which returns a Token associated with the found object (i.e., **PrjTimePath**) and can then be used to set its own variable.

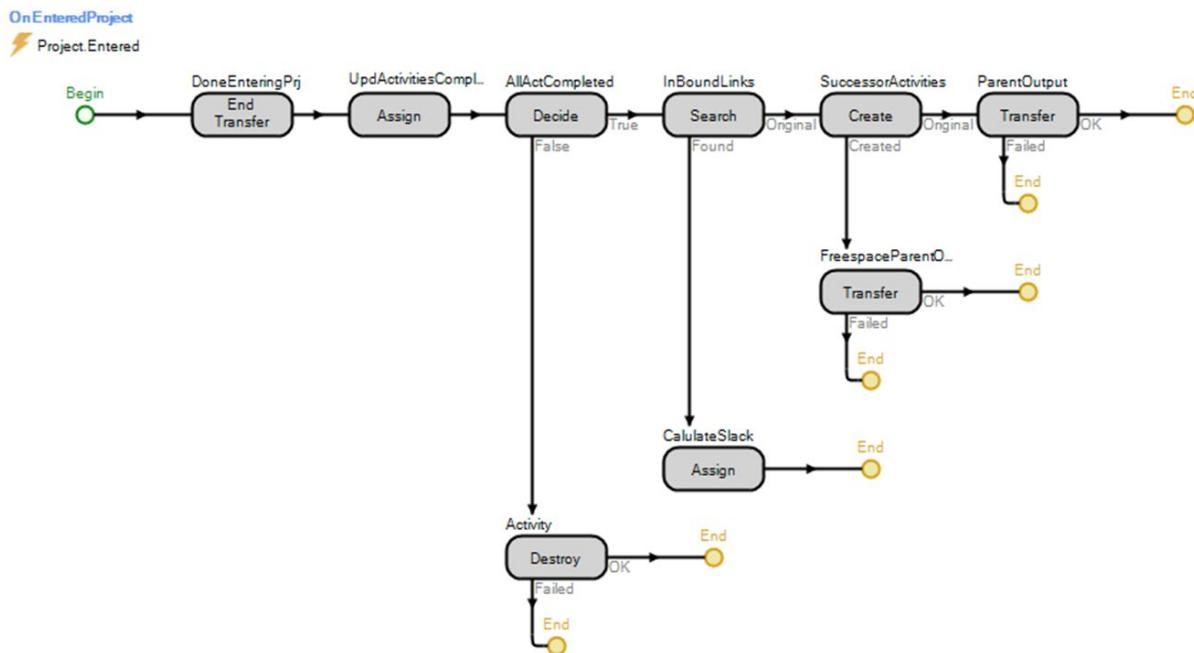


Figure 23.28: Setting the **StaCalculateSlack** Variable of the Paths When All Activities Have Finished

- Figure 23.29 shows the *Search* step that will search a “NodeInBoundLinks” where the *Node Name* property is set to **Input**, which is the parent input node. Note that this node will not be in the dropdown list but will need to be typed into the field. The *Exclusion Expression* (i.e., *JunctionType* ==

`List.ListJunctionType.StartNode`) specifies to skip this *Search* step if this **JUNCTION** is a “StartNode” and does not need to notify in predecessor links.

Properties: InBoundLinks (Search Step Instance)	
Basic Logic	
Collection Type	<b>NodeInboundLinks</b>
Node Name	<b>Input</b>
Search Type	Forward
Match Condition	
Limit	<b>Infinity</b>
Advanced Options	
Starting Index	
Ending Index	
Save Index Found	
Save Number Found	
Copy Over Table Row References	True
Token Wait Action	NewTokensExitFirst
Exclusion Expression	<b>JunctionType == List.ListJunctionType.StartNode</b>

**Figure 23.29: Searching for all Inbound Links**

- Insert an *Assign* on the “Found” branch and specify with the values in Figure 23.28, which will set the `PrjTimePath.StaCalculateSlack` discrete variable to one.

Properties: CalulateSlack (Assign Step Instance)	
Basic Logic	
State Variable Name	<b>PrjTimePath.StaCalculateSlack</b>
New Value	<b>1</b>

**Figure 23.30: Setting the Calculate Slack Variable**

**Step 7:** Save and rerun the experiment.<sup>336</sup>

*Question 5:* What percentage of the time is path A on the critical path?

---

*Question 6:* How long could we delay the start of activity A without affecting the critical path?

---

*Question 7:* How long could we delay the start of activity C and not have any effect on the critical path?

---

**Step 8:** Looking at the results, it is clear that the critical path is B D E F since these paths have the highest percentages of time on the critical path, as seen in Figure 23.31 for the Percent CP average.

<sup>336</sup> Sometimes when adding properties to objects that are already in the model causes ghost errors of things not being specified. Save and reopen the model to eliminate them if this occurs.

Object Type	Object Name	Data Source	Category	Data Item	Statistic	Average	Minimum	Maximum	Half Width
PrjTimePath	A	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)	6.4720	5.0112	9.3587	0.0543
		TallyStatPercentCP	Critical Path	Percent CP	Average	0.0290	0.0000	1.0000	0.0104
		TallyStatSlack	Critical Path	Slack	Average (Days)	2.0403	0.0000	3.8708	0.0576
	B	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)	8.5008	6.5778	8.9992	0.0267
		TallyStatPercentCP	Critical Path	Percent CP	Average	0.9710	0.0000	1.0000	0.0104
		TallyStatSlack	Critical Path	Slack	Average (Days)	0.0116	0.0000	0.8448	0.0048
	C	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)	17.5812	13.6590	23.2258	0.1095
		TallyStatPercentCP	Critical Path	Percent CP	Average	0.0410	0.0000	1.0000	0.0123
		TallyStatSlack	Critical Path	Slack	Average (Days)	3.4945	0.0000	8.5482	0.1119
	D	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)	13.7068	11.0667	15.5315	0.0439
		TallyStatPercentCP	Critical Path	Percent CP	Average	1.0000	1.0000	1.0000	0.0000
		TallyStatSlack	Critical Path	Slack	Average (Days)	0.0000	0.0000	0.0000	0.0000
	E	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)	21.0433	18.4607	23.8164	0.0612
		TallyStatPercentCP	Critical Path	Percent CP	Average	0.9590	0.0000	1.0000	0.0123
		TallyStatSlack	Critical Path	Slack	Average (Days)	0.0324	0.0000	2.3989	0.0123
	F	TallyStatEarliestFinish	Critical Path	Earliest Finish	Average (Days)	33.1002	27.7104	38.1609	0.1153

Figure 23.31: Results of the Critical Path Analysis

### Part 23.7: Adding Slack and Percent Time on CP Calculations Second Approach

The previous section uses the **JUNCTION** objects to signal (i.e., set the **StaSlackVariable**) the activity time paths of the Latest Finish. This section will take the alternative approach and let the **PRJTIMEPATH** monitor a **JUNCTION** state variable. This section demonstrates how an object can monitor the state variables of other objects, which can prove to be very useful in communicating among objects.

**Step 1:** Save the current project under a new name (Chap23-7.spx).

**Step 2:** Remove the *Search* steps from the “*OnEnteringProject*” process and insert a new *Assign* step that assigns the current time (*Run.TimeNow*) to the **StaLatestFinish** state variable.

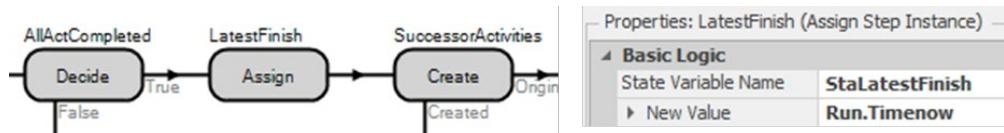


Figure 23.32: Setting the **StaLatestFinish** State Variable

**Step 3:** Next, select the **PRJTIMEPATH** object.

**Step 4:** From the *Definitions→Properties→Add→Standard Property* dropdown, insert a new **STATE** property named **JunctionLF**, which will allow the modeler to specify the **JUNCTION** state variable to monitor. Change the *Category Name* to “Project Management” by specifying it.

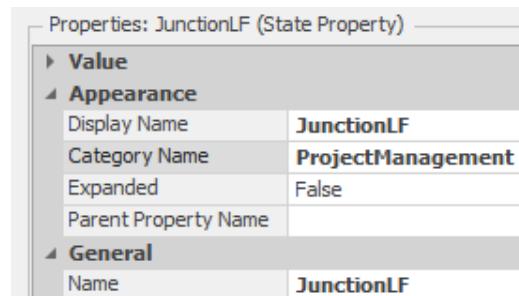


Figure 23.33: Create a State Variable Property

**Step 5:** Modify the **MONITOR** element **MonitorSlack** to monitor the new state reference property **JunctionLF** as seen in Figure 23.34.

Properties: MonitorSlack (Monitor Element)	
Basic Logic	
State Variable Name	JunctionLF
Monitor Type	DiscreteStateChange
Monitored State Variables (More)	0 Rows
Trigger Condition	
Triggered Process Name	MonitorSlack_TriggeredProcessName

Figure 23.34: Using the Reference State Property to Monitor

**Step 6:** Return to the model, and for each **PRJTIMEPATH**, specify the appropriate **JUNCTION**'s **StLatestFinish** state variable.

- For example (see Figure 23.35), the *Junction LF State Variable* property should be set to Junction2.StLatestFinish for both paths **A** and **B**.
- For paths **C** and **E**, **D**, and **F** it should be set to Junction4.StLatestFinish, Junction3.StLatestFinish, and Junction5.StLatestFinish respectively.



Figure 23.35: Specifying the State Variable

**Step 7:** Save the model and rerun the experiment, comparing the results of the two systems.

*Question 8:* Did you get the same results?

---

**Step 8:** The problem was the last path did not calculate the latest finish statistics since it was heading to an “End” junction. Select the **JUNCTION** object and then inside the *OnEnteringProject* process. Recall that all the entities were destroyed via the exclusion property of the *Decide* step (see Figure 23.13). Therefore, it is not sent to the *Assign* step. Therefore, remove the portion of the exclusion property associated with the end node: `2 * (JunctionType == List.ListJunctionType.EndNode)`. Then, add a *Decide* step after the *Assign* to determine if the node is an **EndNode** and send the token to the *Destroy* step, as seen in Figure 23.36.

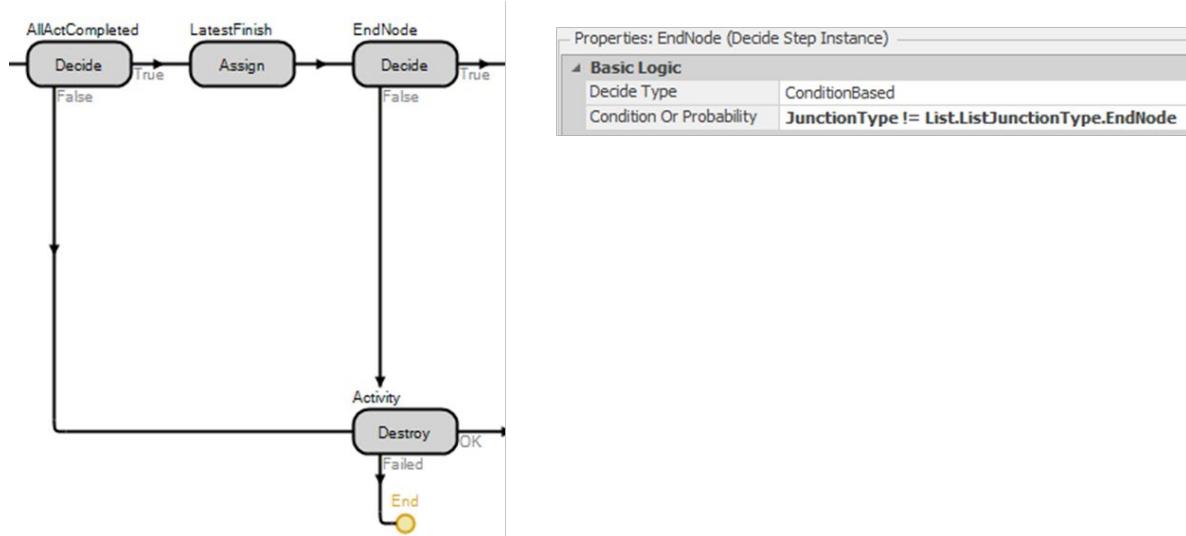


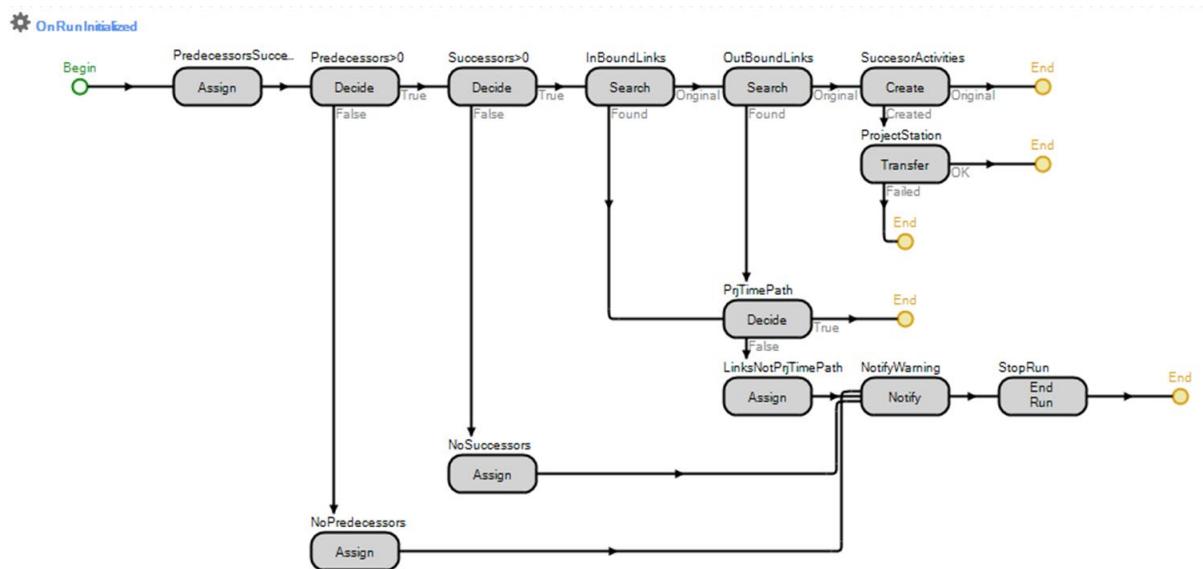
Figure 23.36 Modifying the Exclusion Expression and Adding a Decide Step

**Step 9:** Save the model and rerun just the model.

*Question 9:* Did anything happen when the activity went through the end node?

## Part 23.8: Error Checking to Make Sure Modeler Uses Junction Correctly

There are several assumptions the model has made. We are assuming the modeler will always connect up the junctions correctly using only the new **PRJTIMEPATH** links. To eliminate this assumption, we will modify the **OnRunInitialized** to check for these issues, warn the modeler, and then end the simulation run, as seen in Figure 23.37.



**Figure 23.37: Checking for Valid Links**

**Step 1:** Insert a new STRING STATE variable named **StaWarningMessage** to specify the particular message.

**Step 2:** Insert a conditional-based *Decide* step to check to see if there are predecessor links connected to the Junction. If this is not a “Start Node,” the step will be skipped automatically as specified by the *Exclusion Expression* property. Insert another *Decide* step that will check to make sure the successor links have been defined if this is not an “End Node,” as seen in Figure 23.38.

Properties: Predecessors>0 (Decide Step Instance)		Properties: Successors>0 (Decide Step Instance)	
Basic Logic		Basic Logic	
Decide Type	ConditionBased	Decide Type	ConditionBased
Condition Or Probability	StaPredecessors>0	Condition Or Probability	StaSuccessors>0
Advanced Options		Advanced Options	
Exclusion Expression	JunctionType == List.ListJunctionType.StartNode	Exclusion Expression	JunctionType == List.ListJunctionType.EndNode

**Figure 23.38:** Checking to see if the Predecessors and Successors have been Defined

**Step 3:** If the **Predecessors > 0** *Decide* steps fail (i.e., not start node but links are connected to the junction), insert an *Assign* step on the failed branch that will set the **StaWarningMessage** state variable to a string which states the **JUNCTION** name as well as what to do to fix the issue.<sup>337</sup>

<sup>337</sup> Note the use of `String.Format` function to create string that has one argument (i.e., “{0}”) which is set to the value of the **JUNCTION** Name.

```
String.format("{0} is Not a Start Node and there are no predecessors (i.e., Project Path links) defined", Name)
```

**Step 4:** Next, if the **Successors>0 Decide** steps fail, insert an **Assign** step on the failed branch that will set the **StaWarningMessage** state variable to a string that states the **JUNCTION** name as well as what to do to fix the issue.

```
String.format(" {0} is Not an End Node and there are no successors (i.e., links) defined", Name)
```

**Step 5:** Connect both of the **Assign** steps to the same **Notify** step. The **Notify** step can either write a message in the Trace window or give a pop-up Warning to the user. We will use the **StaWarningMessage** as the **Message Content** property, as seen in Figure 23.39.



Figure 23.39: Sending a Warning to the User

**Step 6:** Insert an **End Run** step that will terminate the simulation since the model has an error, and we don't want to continue.

**Step 7:** The first two **Decide** steps determined if no predecessor or successor links were defined. However, the user could connect the **JUNCTIONS** using normal **PATHS**, **CONNECTORS**, or **TIMEPATHS** which is also an error. Insert two **Search** steps that will search both the inbound and outbound links as seen in Figure 23.40. The inbound search will be skipped if this is a "Start Node" while the outbound search will be skipped if the **JUNCTION** is an "End Node" owing to the **Exclusion Expression** property.

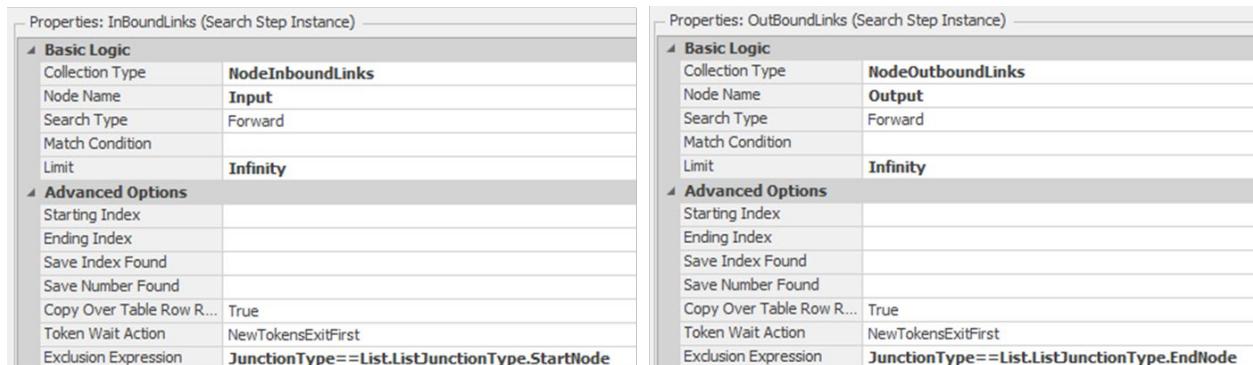


Figure 23.40: Searching to Make Sure the Predecessor and Successor Links are PrjTimePaths

**Step 8:** When links are found, we must check to ensure they are **PRJTIMEPATH** links only. Connect both "Found" branches to a conditional **Decide** step that checks to see if these are **PRJTIMEPATH** using the **Is** keyword, which returns **False** if these objects are not project time paths.

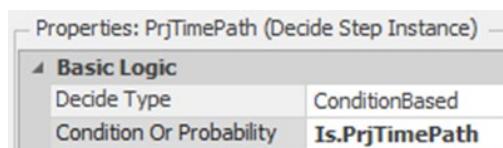


Figure 23.41: Checking to see if the Links are Project Time Paths

**Step 9:** If any of the links are not project time paths, then set the warning message to tell the modeler which link is not a project time path that is connected to which **JUNCTION**, as seen below.

```
String.format("The successor or predecessor {0} connected to {1} is not a PrjTimePaths. Change the link to PrjTimePath",Link.Name, Name)
```

**Step 10:** Next, connect the *Assign* step to the *Notify* step by dragging the End on top of the step.

**Step 11:** Select the Facility window and test the new error checking by inserting another Junction into the model.

**Step 12:** Save and then run the model with the new Junction not connected.

*Question 10:* Does the simulation stop and warn the user of the Junction in error?

---

**Step 13:** Next, connect both the input and output to other nodes via paths and rerun the model.

*Question 11:* Does the simulation stop and warn the user of the Junction in error?

---

## Part 23.9: Commentary

- Sometimes, the activities within a project employ common resources. The resources can limit the number of simultaneous activities, and resource allocation and the timing of the allocation become limits on the project's progress. Common resources can be incorporated directly into the **PRJTIMEPATHS**. However, deciding which activities should be given priority for the resources can become a problem.
- Another possible option in project management is determining the time to delay an activity's start. By delaying the start of an activity, resources can be made available to more critical ones, thus reducing project completion. Optquest could be used to optimally determine the start time.
- One could have also used **TASK SEQUENCES** to model project management problems, but adding all the statistics would have been more difficult.

# Appendix A

## Input Modeling

---

Any discussion of creating a “realistic” model of any operation eventually turns to the issue of variability or randomness. Incorporating “randomness” into your simulation model can be a significant challenge because the causes of randomness are difficult to understand completely. As a result, we approximate this randomness with some kind of “randomness model,” usually a statistical distribution. Because we must “approximate” the randomness, we view this as “modeling.” Because randomness is not explained by the simulation model, it must be “input” to the simulation. Thus, the reason for the term “Input Modeling.” There are several forms of input models for randomness.

One input model is the time-based arrival process, which in SIMIO is called the “Time Varying Arrival Rate” arrival mode at a SOURCE. Fundamentally, we are trying to model an arrival process that changes throughout time. The simplest model is a Poisson arrival process whose mean changes with time, which is generally referred to as a Non-Homogeneous Poisson Process (NHPP). Recall that the Poisson has only one parameter: its arrival rate (mean). A NHPP has a Poisson mean with an arrival rate change that can be any function of time. In SIMIO, the rate function is a stepped function that provides an arrival rate during a specified time period. In SIMIO, the rate is always given in events per hour, while the time periods have a fixed interval size (and number). A more sophisticated NHPP would permit a more flexible rate function.

Another model of randomness is the specification of a statistical distribution for some observed randomness, usually an amount of time. The most prominent examples are Processing Times at SERVERS, WORKSTATIONS, COMBINERS, and SEPARATORS and Travel Time for TIMEPATHS. Other examples of times include Transfer-In Time, Setup Time, and Teardown Time in various objects. Also, the “Calendar Time Based” failure type at working objects requires uptime between failures and time to repair. Creating input models for this kind of input may employ some way to find a statistical distribution to match the randomness behavior. Sometimes, there is randomness in a number, such as the size of an arriving batch or the number of parts in a tote pan.

### Part A.1 Random Variables

Randomness is thus represented in simulation by random variables. These random variables describe either a continuous value (such as time) or a discrete value (such as a quantity). While you may not know how the randomness occurs, you might know something about the distribution of these values, such as some descriptive statistics. Descriptive statistics include measures of central tendency such as a mean, median, or mode. Maybe it’s a measure of variability such as a variance, standard deviation, coefficient of variation, or range. Other descriptive statistics include the measure of asymmetry called the skewness and the measure of the peakedness called the kurtosis.

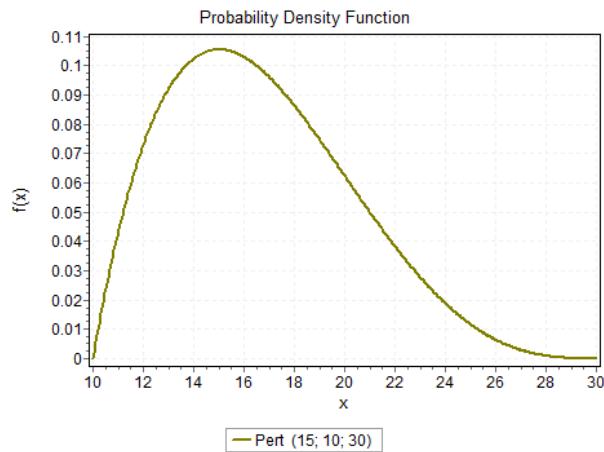
### Part A.2 Collecting Data

More than likely, you don’t know much and will need to collect some data from the real input process to get some idea of how the randomness behaves. This data is simply a “sample” from which you might want to estimate how the population is distributed.

However, before we begin to examine how to use “sample data,” you need to remember the most important rule of data collection: *Build the simulation model before collecting data!*

Collecting data is almost always an onerous, frustrating, and expensive process. You must know that you need the input model before you mount a data collection effort. The best way to know that you need particular data is to construct the simulation model first. The simulation model doesn't know if the input models were obtained from real data. Simply use your best judgment to obtain input models.

One suggestion is to interview some people who are familiar with the process that is generating the randomness and try to elicit some general characteristics from them. If you can only estimate a minimum and a maximum, use a uniform distribution to match. However, generally, you can obtain three estimates: a mode, a minimum, and a maximum. For example, suppose we want to estimate the randomness that describes the amount of time it takes you to get to work or school. If I say, tell me the most likely or most common time it takes, and you say 15 minutes. That estimate of 15 minutes becomes a mode. Now tell me how long it takes to make all the lights and cruise here without difficulty, namely the minimum amount of time. Suppose you say 10 minutes. Next, give me the maximum when everything goes wrong. You say the worst is 30 minutes. With these estimates, a PERT distribution (sometimes called the BetaPERT) is used, as shown in Figure A.1.



**Figure A.1: Pert Distribution**

As you can see, this distribution ensures that our randomness is bound to realistic values. In this case, there are no negative values<sup>338</sup>, the distribution is usually skewed to the right, and the mode is less than the mean. We highly recommend this means of obtaining estimates, especially in the early modeling stages. The Triangular distribution can also be used to model the case when given the minimum, maximum, and most likely points. However, it will be heavier in the tails, while the Pert will have more probability around the mode.

With the liberal use of these kinds of estimates, build your simulation model. Vary some of the input model parameters (i.e., the mode, minimum, or maximum) and see the impact they have on the simulation outputs. In particular, note how they impact the simulation outputs that are important to you. With a little experimentation of this type – called a “sensitivity analysis,” you begin to get a sense of which data are most critical. You can marshal a data collection effort proportional to the input model’s importance. There are a number of hard-to-answer questions regarding data collection. Often, you can’t collect as much data as you

<sup>338</sup> If a sample from a distribution for a time, like a processing time, produces a negative value, SIMIO will display an error. Distributions that can have negative values, like the Normal must be used with care. You can avoid negative values using and expression like `Math.Max(0.0, Random.Normal(10.0,4.0))`.

would like, so you will need to consider the limitations of your data collection budget. You will want to collect as much data as you can, bearing in mind that not all data is equally important in the model.

### Part A.3 Input Modeling: Selecting a Distribution to Fit Your Data

Once you begin the data collection effort, you can begin to select distributions that fit your data. Why do we want to select a distribution to fit the data and not simply use the data directly in the simulation model? First, the data is only a sample from a population and not the total population, so we don't know precisely whether the sample completely represents the population from which it is sampled. Secondly, real data tends to be incomplete in that it doesn't completely match the process being modeled. Third, we believe that there are enough standard statistical distributions from which to choose that will do a good job of modeling the "true" input process. So, we will look for a distribution to fit the data.

Fortunately, once we have some data, the process of finding a distribution to fit the data is somewhat routine. Here are the steps we will follow:

- 1) Collect data and put it into a spreadsheet
- 2) Identify candidate distributions
- 3) For each candidate distribution
  - a) Estimate parameters for the candidate distributions
  - b) Compare candidate distribution to histogram
  - c) Compare the candidate distribution function to the data-based distribution function
- 4) Select the best representation of the input (the input model)

#### Candidate Distributions

Before going into much detail, it is useful to do a SIMIO "Help" on "distributions." You will see descriptions of all the distributions that SIMIO supports. Presently, they include the standard "continuous" distributions of Beta, Erlang, Exponential, Gamma, JohnsonSB, JohnsonSU, LogLogistic, LogNormal, Normal, PearsonVI, Pert, Triangular, Uniform, and Weibull as well as the standard discrete distributions of Binomial, Geometric, NegativeBinomial, and Poisson. Also, there are empirical distributions called continuous and discrete, which represent data-based continuous and discrete distributions. You should spend some time looking through these distributions, so you have some sense of their "shape" and parameters. For instance, Figure A.2 displays the Erlang from the SIMIO help.

##### Random.Erlang(mean, K)

The Erlang distribution models an n-phase activity where the time for each phase is exponentially distributed. The Erlang has parameters that specify the mean and number of integer phases (K).

Erlang is the sum of K exponentials, each with an expected value of Mean/K. The Mean parameter is the mean of the Erlang, not the mean of the individual exponentials.

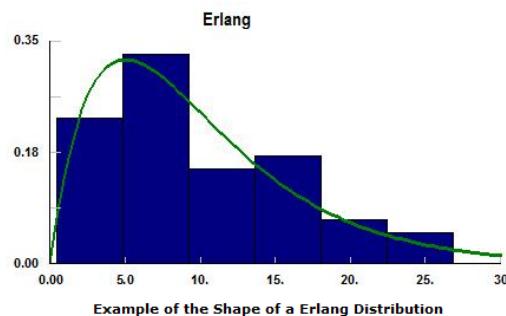


Figure A.2: Erlang Distribution

The display shows that Erlang is a non-negative distribution specified by a mean and a number of phases (each phase being Exponentially distributed with a mean of the distribution divided by the number of phases).

In the prior figure, a histogram is displayed. A histogram is simply a display of the frequency in which data values fall within a given range. Typically, the ranges or cells of a histogram have identical ranges (sometimes the end cells are longer), and the number of histogram cells ranges from 5 to 15. The histogram is an arbitrary display since the user determines the size and number of cells, but some “rules of thumb” exist. For example, by dividing the range of observations (i.s., maximum – minimum) by the number of cells, you can determine the range for each cell. Although you would like to keep the number of cells, C, between 5 and 15, the following can be a good guide:

$$\text{Scott's Rule: } C \cong \frac{5}{3}(n^{\frac{1}{3}})$$

$$\text{Surges'Rule: } C \cong 1 + 3.3 \log_{10}(n)$$

Here, the number of observations in your data is  $n$ . With the range for each cell, you simply display the portion of the observations that fall within each range. By examining the histogram, you may want to select certain distributions to fit. But before you fit a distribution to the data, you need the parameters for the distribution, and they will need to be estimated from the data.

### Estimating Distribution Parameters

The estimation of distribution parameters employs one of two methods: moment match and maximum likelihood. Moment matching requires that you know, for a two-parameter distribution, the mean and variance of the distribution in terms of its parameters. If the Erlang mean is given by the symbol  $\theta$  and there are  $k$  phases to the Erlang, then the

$$\text{Mean} = \theta$$

$$\text{Variance} = \theta^2/k$$

Therefore if you compute the average and the sample variance from your data as  $\bar{x}$  and  $s^2$ , approximate mean by the average and the variable by the sample variance. Solving, you can approximate the parameters of the Erlang by the following equations

$$\theta = \bar{x}$$

$$k = (\bar{x}/s)^2$$

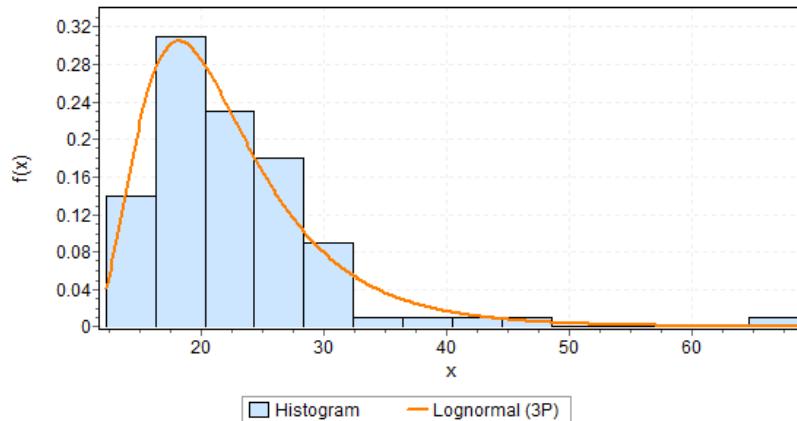
The method of “matching” the sample statistics to the parameter values is the essence of moment matching.

Maximum likelihood requires more statistical development than we wish to devote here. However, the basic idea is to find the parameters of the distribution so they have the greatest “likelihood” of coming from that parameterized distribution. Most statistical books that deal with “estimation” contain a description of maximum likelihood. In actual practice, the process of finding the maximum likelihood can become a computational challenge if done by hand, so using some kind of software to perform the computations is highly desirable. We do need to note that most statisticians prefer the use of maximum likelihood-to-moment matching for parameter estimation. The reasons are somewhat technical, but the general maximum likelihood

will have the best parameter estimates (they are optimal in the sense that they produce the best unbiased, minimum variance estimates).

## Determining Goodness-of-Fit

Once a candidate distribution has been identified and its parameters determined, you can now compare the distribution you fit to the data. Here is an example of a histogram with a candidate distribution (Lognormal) being displayed in Figure A.3. The distribution parameters have been computed from the data.

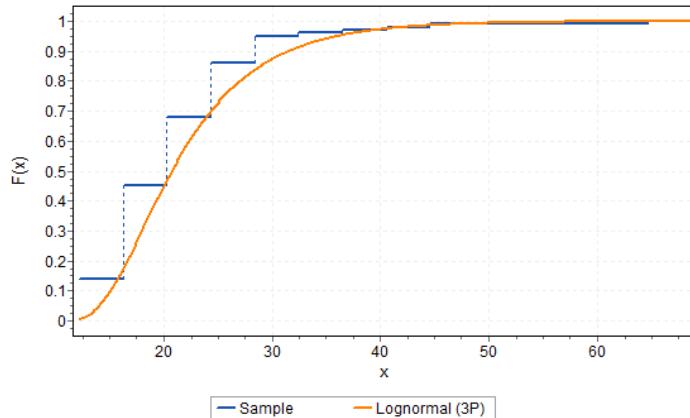


**Figure A.3: Probability Density Function**

The basic question is how good is the “fit” of the Lognormal to this data? Our answer is based on two approaches: visual inspection and statistical tests.

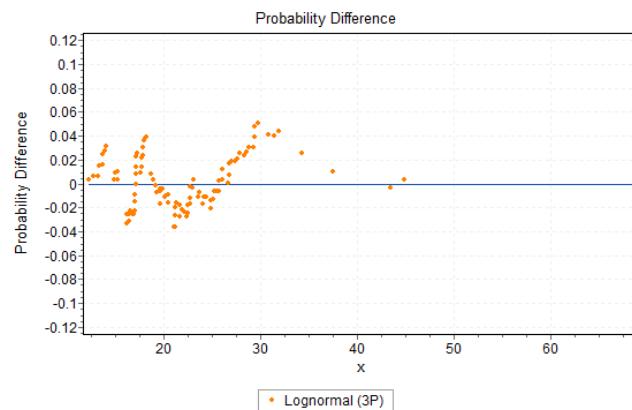
**Visual Inspection:** Visual inspection is the most important strategy because it visualizes the fit. Looking at the prior histogram and the fitted distribution, it’s most important to see how the fitted distribution matches the observed histogram, particularly in the tails. The “tails” are most important because the variation in the distribution is largest in the tails, and the tails most generally affect the queuing, utilization, and flow time statistics.

Several other ways exist to “see” how well the fitted distribution matches the data. Most are based on the cumulative distribution function (CDF). For example, the comparison of the fitted CDF versus the data-based CDF is given in Figure A.4.



**Figure A.4: Cumulative Distribution Function**

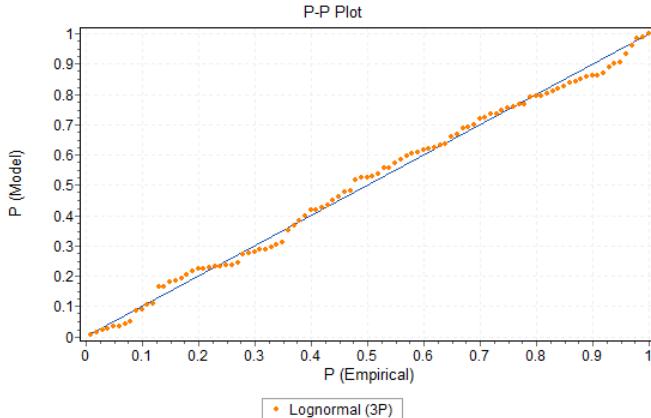
Again, we looked at how well the fitted distribution represents the data. When looking at the difference, it is useful to plot the difference in the two CDFs as in Figure A.5



**Figure A.5: Difference between Empirical CDF and Fitted CDF**

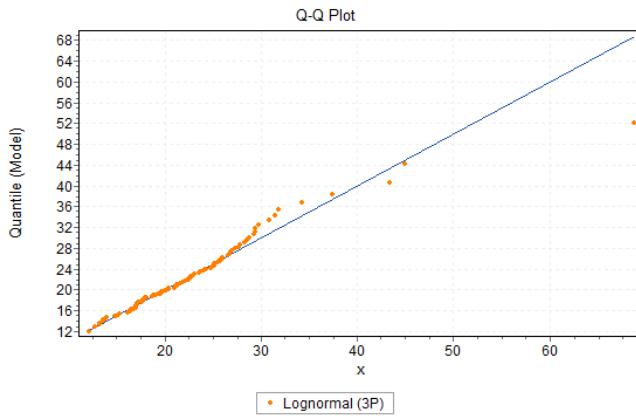
In this plot, we are looking to see if the difference is generally above or below 0, since that would indicate that the fit overestimates or underestimates the data.

Two of the more useful visual plots are based on the “distance” between the fitted CDF and the data-based CDF. If you measure the difference vertically, you obtain the PP plot, which emphasizes the difference in the center of the distribution. The PP plot is shown in Figure A.6.



**Figure A.6: P-P Plot Example**

If you measure it horizontally, you obtain the QQ plot, which emphasizes the difference in the tails of the distribution. The QQ plot is shown in Figure A.7.



**Figure A.7: Q-Q Plot Example**

In both the PP and QQ plots, we want to see the points “close” to the dividing line. Deviations from the line indicate a potential for overestimation or underestimation of the data. Finally, we tend to have more interest in the QQ plot than in the PP plot since it emphasizes the difference in the tails of the distribution, where we believe the fit is most important.

**Statistical Tests:** Statistical tests are based on classical statistical testing in which a null hypothesis is compared to an alternative. The null hypothesis is that the data comes from the fitted distribution. However, the “significance” of a statistical test is when the null hypothesis is rejected (Type 1 error). However, in goodness-of-fit, we want to accept the null hypothesis so the power of the test is relevant (Type 2 error); hence, we would like a larger p-value. Thus, statistical tests tend to reject only the poorest of fitted distributions. It is the reason we place so much emphasis on visual inspection.

There are three standard goodness-of-fit statistical tests: the Chi-Squared test, the Kolmogorov-Smirnov test, and the Anderson-Darling test. These tests are more thoroughly described in statistical texts. However, we can provide a general description of each.

The Chi-Squared test is based on comparing the observed number of observations in a histogram cell with the number expected if the fitted distribution was the input model. Although the test is often based on equal cell widths, it is generally better to use equal probability cells for the test.

The Kolmogorov-Smirnov (K-S) is a non-parametric test that looks at the largest (vertical) distance between the fitted CDF and the data-based CDF. The test is only approximate if parameters have to be estimated (which they usually are). A generalization of the K-S test is the Anderson Darling, which puts more weight in the tails of the difference (the K-S uses equal weights).

## **Part A.4 Distribution Selection Hierarchy**

The following is a rule of thumb and represents the distribution selection hierarchy one should follow.

1. Use recent data to fit the distribution using input modeling software
2. Use raw data and load discrete points into a custom distribution (i.e., Empirical CDF)
3. Use the distribution suggested by the nature of the process or underlying physics
4. Assume a simple distribution and apply reasonable limits when lacking data (e.g., Pert or Triangular)

It is always better to fit distributions to data collected about the process. If no good fits can be achieved, the empirical distributions can be used to model the data. If some underlying distribution is suggested about the process owing to scientific processes, then one can utilize these. Assuming a simple distribution and applying reasonable limits can be used as a start if you have very little or no data. Often, we can obtain the minimum, maximum, and sometimes the most likely. In this case, we can use the Pert distribution.

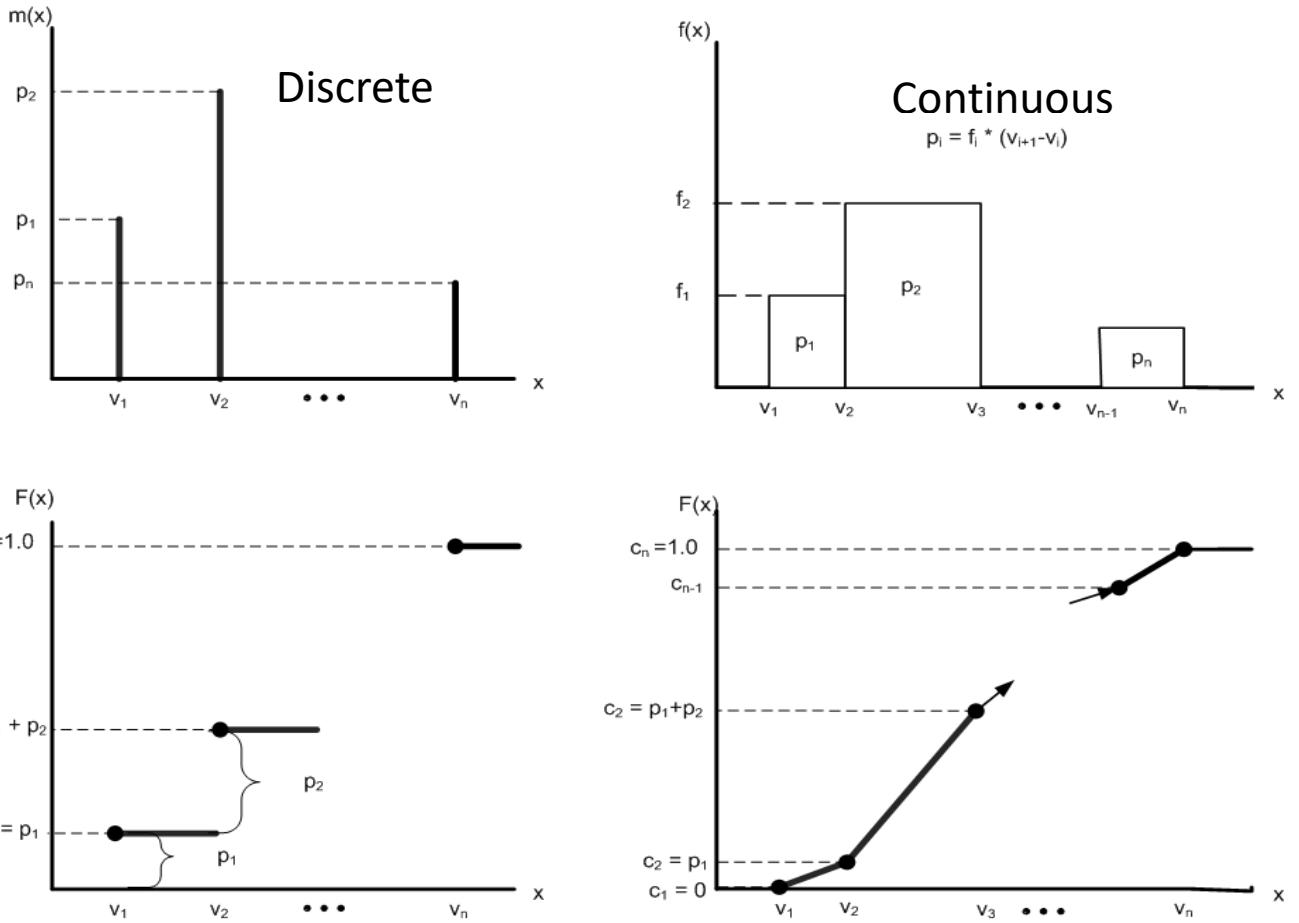
## **Part A.5 Empirical Distributions in SIMIO**

In contrast to the standard distributions, for which a functional form must be determined, parameters must be estimated, and empirical distributions are based totally on your data. You have two choices for empirical distributions: discrete and continuous. These distributions are represented in Figure A.8, which describes the probability mass/density function as well as the cumulative distribution function.

The SIMIO statements for the distributions in Figure A.9 and Figure A.10 are specified according to the following cumulative distributions:

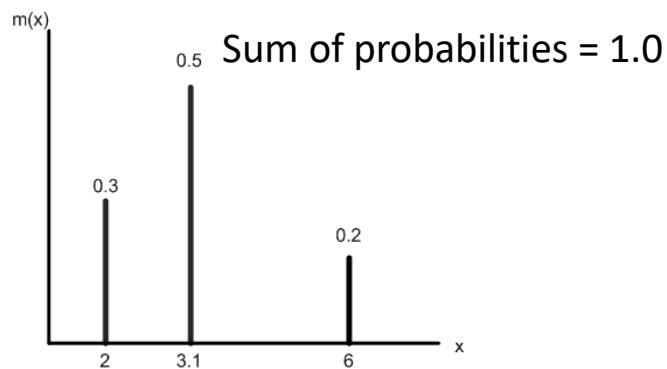
```
Random.Discrete (v1, c1, v2, c2, ..., vn, cn)
```

```
Random.Continuous (v1, c1, v2, c2, ..., vn, cn)
```



**Figure A.8: Empirical Distributions in SIMIO**

Of the two distributions, the Discrete is the most often used. We might have a case where 30% have a value of 2.0, 50% have a value of 3.1, and 20% have a value of 6. A plot of this case is shown in Figure A.9.



**Figure A.8: SIMIO specification:** `Random.Discrete(2, 0.3, 3.1, 0.8, 6, 1.0)`

Some other examples of the use of the empirical discrete distribution in SIMIO are:

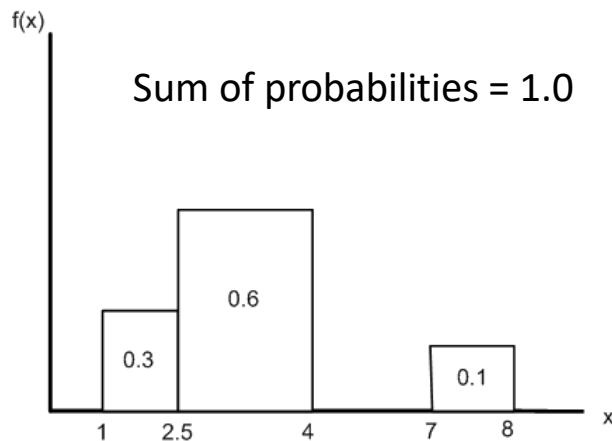
`Random.Discrete(2, 0.3, Math.Epsilon +5.2, 0.8, 6, 1.0)`

```
Random.Discrete(Random.Exponential(0.5), 0.3, Run.TimeNow, 0.8, 6, 1.0)
```

```
Random.Discrete("Red", 0.3, "Green", 0.8, "Blue", 1.0)
```

Of course, the value/expression can be negative, although it occurs infrequently. The cumulative probability parameters/expression must also be in the range 0 to 1, and the last cumulative probability parameter must be 1.

An example of continuous distribution might be where 30% of the time, the value falls (uniformly) between 1 and 2.5, 60% between 2.5 and 4, and 10% between 7 and 8. The plot for this case is shown in Figure A.10.



**Figure A.9: SIMIO Specification:** Random.Continuous(1, 0, 2.5, 0.3, 4, 0.9, 7, 0.9, 8 1.0)

The values/expression should not decrease, although SIMIO does not check. Also, cumulative probability parameters/expression values must be in the range 0 to 1, and the last cumulative probability parameter must be 1.

## Part A.6 Software for Input Modeling

The computational aspects of the distribution-finding process have been incorporated into the software. Several software “packages” exist now to help in fitting a distribution to data (SAS Jmp, ExpertFit, EasyFit, Minitab). The software will do the work of finding candidate distributions that represent the data.

Typical of input modeling software is *EasyFit* from <http://www.mathwave.com>. It comes with a free trial version, and a variety of licenses may be obtained. The software has a catalog of over 50 standard distributions. Data can be read from worksheets or entered and edited in the software’s worksheet. The fitting system considers all the distributions as possible candidates. However, The user may limit the fitting to (continuous) distributions with an upper or lower bound or both and whether these bounds are open or closed. Maximum likelihood estimates are used for distribution parameter estimation for each candidate distribution. The visual displays include the probability density (or mass) function, the cumulative distribution function, the PP and QQ plots, and the CDF difference plot. The histogram of the data can also specify the number of bins (cells). The statistical goodness-of-fit tests include the K-S, the Anderson-Darling, and the Chi-Squared with equal probability or equal width cells. The candidate distributions can be ranked by the statistics produced from the statistical tests.

Whenever you use input modeling software, you must be concerned with parameter correspondence between the software parameters and the SIMIO parameters. For example, the EasyFit parameters of Erlang ( $m, \beta$ ) correspond to Random.Erlang ( $m * \beta, m$ ) in SIMIO. Table A.1 shows how to convert EasyFit parameters to SIMIO parameters.

**Table A.1: Easy Fit to SIMIO Conversion of Input Parameters**

Distribution	Easy Fit	SIMIO
Bernoulli	Bernoulli(p)	Random.Bernoulli(p)
Beta	Beta( $\alpha, \beta$ )	Random.Beta( $\alpha, \beta$ )
Binomial	Binomial(nt,p)	Random.Binomial(p,n)
Erlang*	Erlang(m, $\beta$ )	Random.Erlang( $m * \beta, m$ )
Exponential*	Exponential( $\lambda$ )	Random.Exponential( $\lambda$ )
Gamma*	Gamma( $\alpha, \beta$ )	Random.Gamma( $\alpha, \beta$ )
Geometric	Geometric(p)	Random.Geometric(p)
Johnson SB**	JohnsonSB( $\gamma, \delta, \lambda, \xi$ )	Random.JohnsonSB( $\gamma, \delta, \xi, \xi + \lambda$ )
Johnson UB(SU)**	JohnsonSU( $\gamma, \delta, \lambda, \xi$ )	Random.JohnsonUB( $\gamma, \delta, \xi, \lambda$ )
Log-Logistic*	Log-Logistic( $\alpha, \beta$ )	Random.LogLogistic( $\alpha, \beta$ )
Lognormal*	Lognormal( $\sigma, \mu$ )	Random.Lognormal( $\mu, \sigma$ )
NegativeBinomial	Neg.Binomial(n,p)	Random.NegativeBinomial(p,n)
Normal	Normal(stDev,mean)	Random.Normal(Mean,stDev)
PearsonVI*	Pearson6( $\alpha_1, \alpha_2, \beta$ )	Random.PearsonVI( $\alpha_1, \alpha_2, \beta$ )
Pert	Pert(m,a,b)	Random.Pert(a,m,b)
Poisson	Poisson( $\lambda$ )	Random.Poisson( $\lambda$ )
Triangular	Triangular(m,a,b)	Random.Triangular(a,m,b)
Uniform	Uniform(a,b)	Random.Uniform(a,b)
Weibull*	Weibull( $\alpha, \beta$ )	Random.Weibull( $\alpha, \beta$ )

\*---Distributions that have a “Location” parameter but were not included in the syntax. For an example of how to use a location parameter in a SIMIO random variable distribution, see below.

\*\*---Distributions whose “Location” Parameters were indicated in the syntax to show how to derive some of the parameters in SIMIO of the same distribution.

For many distributions, EasyFit generates “location” parameters along with the distribution parameters. This location parameter permits a distribution to be located at some arbitrary point. If EasyFit provides a location, say  $\delta$ , for the Erlang, then you will form the SIMIO expression:

$$\delta + \text{Random.Erlang}(m * \beta, m)$$

The location parameter is used to shift a distribution so it will not be based at zero.

### Part A.7 The Lognormal Distribution

The Lognormal Distribution may require some explanation in case you want to use it. The Lognormal is defined as the distribution you get by taking the logarithm of a normally distributed random variable. That

normal distribution defined with a mean  $\mu$  and a standard deviation  $\sigma$  is, in fact, the parameters of the SIMIO and the EasyFit Lognormal distribution. However, they are not the mean and standard deviation of the Lognormal distribution, which is given by:

$$\text{LogMean} = e^{\mu + \sigma^2/2}$$

$$\text{LogStd} = \sqrt{e^{2\mu + \sigma^2} (e^{\sigma^2} - 1)}$$

If you have data from which you compute  $\bar{x}$  and  $s_x$ , then you can compute SIMIO parameters (the Normal distribution parameters) from:

$$\mu \approx \ln(\bar{x}^2 / \sqrt{\bar{x}^2 + s_x^2})$$

$$\sigma \approx \sqrt{\ln(1 + s_x^2 / \bar{x}^2)}$$

### Part A.8 Modeling the Sum of $N$ Independent Random Variables

We are often faced with determining the processing time for a batch of parts to be processed individually. The processing time is, therefore, a sum of  $N$  independent random variables. For example, if there were 25 parts in a batch and each one took a  $\text{Triangular}(a, m, b)$  minutes to be processed, then the total time to process the batch is given by the summation of 25 independent triangular distributions as seen in the following equation.

$$\text{Processing Time} = \sum_{i=1}^{25} \text{Triangular}(a, m, b)$$

Often, people will mistakenly model the processing time of summation of  $N$  independent random variables as  $N$  times the random variable of interest, which is incorrect as it does not correctly create the appropriate distribution, but it is easy to specify.

$$25 * \text{Triangular}(a, m, b) \neq \sum_{i=1}^{25} \text{Triangular}(a, m, b)$$

To illustrate the point, let's take the Triangular distribution given a minimum value of  $a$ , mode of  $m$ , and a maximum value of  $b$ . The mean and variance of this distribution are given in the following table.

Mean	$\frac{a + m + b}{3}$
Variance	$\frac{a^2 + m^2 + b^2 - am - ab - bm}{18}$

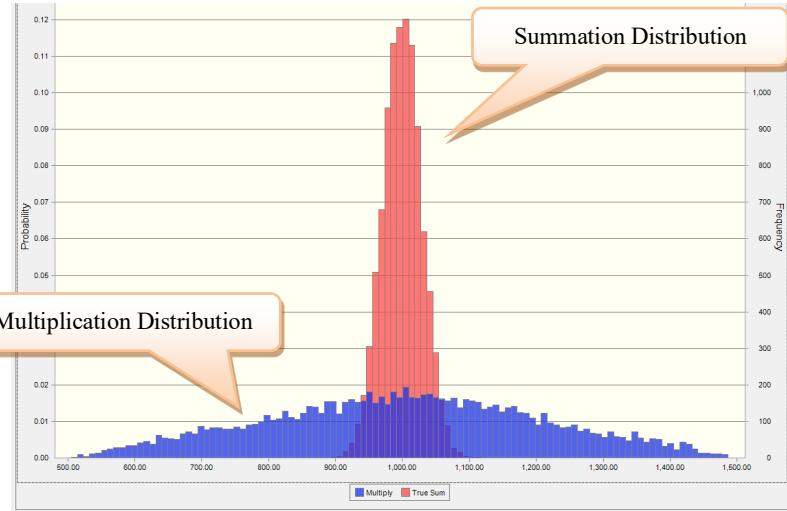
Because it is much easier to perform multiplication rather than summation, people will utilize the multiplication form given by the following equation.

$$N * \text{Triangular}(a, m, b) = \text{Triangular}(Na, Nm, Nb) \neq \sum_{i=1}^N \text{Triangular}(a, m, b)$$

As one can see in Table A.2 below, the means of the two different versions are the same. However, the variances are not. In other words, the modeling distribution is not what is desired. Figure A.11 shows the PDF of each distribution; one can see that the true underlying distribution (i.e., summation distribution) is quite different. The reason is the variances of the summation of  $N$  independent variables add together rather than the standard deviations.

**Table A.2: Calculation of the Mean and Variance of Summation and Multiplication Methods**

Summation Mean	$\sum_{i=1}^N \left( \frac{a + m + b}{3} \right) = N \left( \frac{a + m + b}{3} \right)$
Summation Variance	$\sum_{i=1}^N \left( \frac{a^2 + m^2 + b^2 - am - ab - bm}{18} \right) = N \left( \frac{a^2 + m^2 + b^2 - am - ab - bm}{18} \right)$
Multiplication Mean	$\left( \frac{(Na) + (Nm) + (Nb)}{3} \right) = N \left( \frac{a + m + b}{3} \right)$
Multiplication Variance	$\begin{aligned} & \left( \frac{(Na)^2 + (Nm)^2 + (Nb)^2 - (Na)(Nm) - (Na)(Nb) - (Nb)(Nm)}{18} \right) \\ & = N^2 \left( \frac{a^2 + m^2 + b^2 - am - ab - bm}{18} \right) \end{aligned}$



**Figure A.10: Comparing True Summation Distribution with the Multiplication Distribution**

### Correct Handling of Summation of $N$ Independent Variables

The most appropriate method is to sample  $N$  random variables and then sum these values to reach the correct sample is to use the `Math.SumOfSamples(randomExpression, numberofSamples)` function. You specify the random distribution (i.e., random Expression) and the number of samples needed.

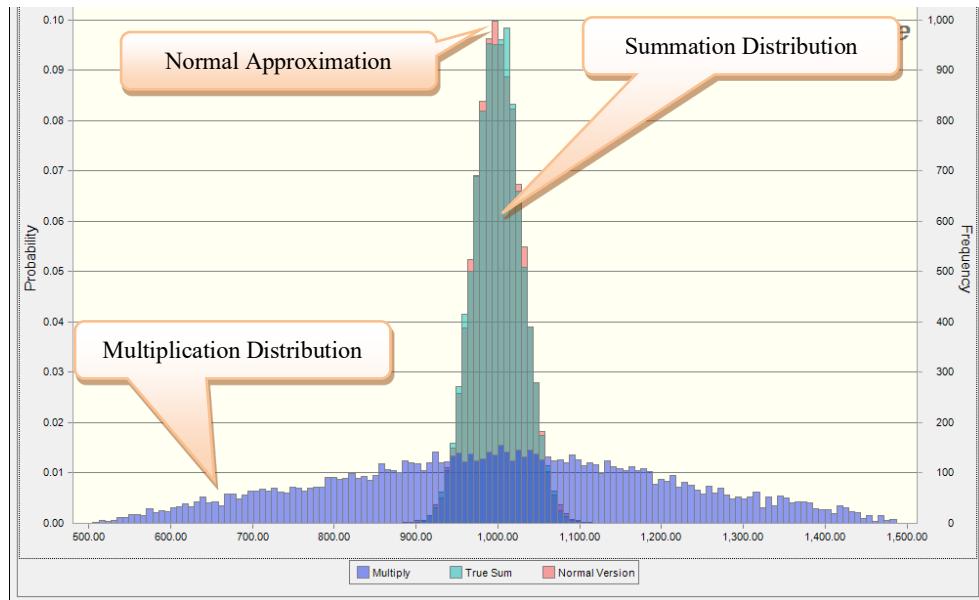
```
Math.SumOfSamples( Random.Triangular(10,15,20), 25 )
```

### Approximation for the Summation of $N$ Independent Variables

As  $N$  gets large, the distribution that results from the summation of  $N$  independent distributions becomes Normal owing to the *Central Limit Theorem*. Therefore, given a large enough  $N$ , the summation distribution can be approximated using a Normal distribution with the mean and standard deviation (STD) in Table A.3. Figure A.12 shows the exact summation distribution, the normal approximation, and the incorrect multiplication distribution. As you can see, the approximation is very good, given  $N$  was 25.

**Table A.3: Calculation of the Mean and Variance of Summation and Multiplication Methods**

Normal Mean	$N * \text{Single Distribution Mean}$
Normal STD	$\sqrt[2]{N * \text{Single Distribution Variance}} = \sqrt[2]{N} * \text{Single Distribution STD}$



**Figure A.11:** Simplified Normal Approximation versus the True Summation Distribution

The disadvantage of using the normal approximation is that the sampled values can become negative and the value  $N$  needs to be sufficiently large. Depending on the distribution,  $N < 10$  will not yield a very good normal approximation, and typically,  $N$  larger than 20 is sufficient.

# Index

---

## A

### *Add Additional Symbol*

ModelEntity, 90, 96, 152, 268, 332

  Animated People, 34

  People, 277

  Vet Patients, 238

  Resource, 325, 338, 381

Add-On Process Triggers. *See* Process Add-On Process Triggers

  Double Clicking to Create, 106

Allocation

  Queue, 254, 312, 313, 391

Allocation Queue

*Animated*, 339

Animated Queue, 144

  Parking Station, 214

Animation, 32

  Circular Gauge, 382

  Status Labels, 172, 381, 404

    ModelEntity, 249, 353, 412

  Status Pie, 160

  Status Plot, 381

  Status Tables, 172, 179

Animated Queue

  Alignment, 139

Appointment Schedules, 140

  Arrival Events Per Time Slot, 142

  Arrival No-Show Probability, 142

  Arrival Time Deviation, 142

  Arrival Time Property, 142

  Data Table, 140, 150

  Entities Per Arrival, 142

  Repeat Arrival Pattern, 142

Arrival Mode. *See* Source Arrival Mode

AssociatedStation Functions, 258

  Capacity, 258

AssociatedStationload, 131

AssociatedStationLoad, 258

AssociatedStationOverload, 258

## B

BasicNode, 42

  Converting to TransferNode, 215

  Definition, 66, 70

  External View – Input Node, 327, 404

Batch Members Queue. *See* Model Entity Batch Members

## C

Candidate, 94

  .Entity.Priority, 266, 338

  .ls, 284

  .ModelEntity.ID, 312

  .Node.AssociatedStationLoad, 131

.Node.NumberRoutingIn, 94

.Resource.Failure, 236

.Server.Inputbuffer.Contents, 94

.Server.Processing.Contents, 94

.Worker.ID, 273

Definition, 273

Dynamic Selection Rule, 266

Search Match Condition, 252, 283

Changeovers, 159

Combiner, 91

  Match Members, 92

Connector, 66

ContainerEntity, 394

Conveyor, 225, 324

  Accumulating, 225

  Non-Accumulating, 225

  Stopping and Starting, 227

Costing. *See* Financials

Create Symbol, 378

## D

Data Table. *See* Tables Data Table

Debugging

  Break Point Button, 270

Detached Queue, 139, *See* Animated Queue

Draw Queue, 95

Drawing

  Create Symbol. *See* Create Symbol

Dynamic Labels. *See* Labels

Dynamic Selection Rule

  Server, 145

## E

Elements, 102

Failure

  Definition, 340

Material, 164

Monitor, 379, 431

Output Statistic, 147, 307

Process, 363

RoutingGroup, 348

State Statistic, 177

Station, 340, 401, 424

  Defintion, 340

Storage, 138, 331

Tally, 143

Tally Statistic, 109, 111, 177, 420

Timer, 167, 177, 315

  Definition, 341

Emptier, 394

Ending Time. *See* Simulation: Ending Time

Event, 388

  Creating New, 389

  Fire, 390

Monitor, 169, 380  
Station Events  
    Entered, 360, 425  
    Exited, 365  
Timer, 167, 181, 408  
TransferNode Events, 349  
Wait for, 389  
Zero-Time, 270, 344  
Experiment, 43, 308  
    Input Analysis, 47, 204  
        Tornado Chart, 47  
        Tornado Chart, 206  
KN Select Best Scenario. *See* KN Select the Best Scenario  
Pivot Grid, 45, 430  
Raw Data, 46  
Reports, 45  
Response Chart, 185  
Responses, 45, 183  
Scenarios, 44, 147, 183  
SMORE, 46, 184  
    Explained, 148  
Subset Selection, 186  
Warm-up Period, 206  
Expression Editor, 30  
*External View*, 358  
    InputNode, 327, 404  
    Junction, 424  
    OutputNode, 327, 404  
    SubModel, 326  
    Visible Button, 402

## F

Failure, 229  
Processing Count Based, 130  
Types  
    Calendar Time Based, 127  
    Event Count Based, 127  
    Processing Count Based, 127  
    Processing Time Based, 127  
Types Explained, 127  
Filler, 394  
Financials, 318  
    Cost Properties  
        Fixed Objects Explained, 319  
        **Model Entities Explained**, 319  
        **Worker and Vehicle Explained**, 319  
Used, 319  
Flow Connector, 382  
Flow Node, 382  
    Initial Maximum Flow Rate, 384  
    Regulator Initially Enabled, 384  
Flow Regulator, 390  
    CurrentMaximumFlowRate, 390  
    Enabled, 390  
Flow Sink, 382, 384  
Flow Source, 382, 383  
FlowContainer  
    Contents.Volume, 384  
    Contents.Volume.Rate, 384

FlowToItemConverter, 394

## G

Google 3D Warehouse, 238

## H

Half-width  
    Estimating Number of Replications, 44

## I

Input Analysis. *See* Experiment: Input Analysis  
Input Sensitivity. *See* Experiment: Input Analysis  
Is, 287  
    Entity, 144  
    ModelEntity, 287  
IsParked, 292  
ItemstoFlowConverter, 394

## K

KN Select the Best Scenario Algorithm, 186, 191, 322

## L

Labels  
    Dynamic Labels, 55  
Link  
    Conveyor. *See* Conveyor  
    Path. *See* Path  
    Selection Weight, 125  
    TimePath. *See* TimePath  
List, 158  
    Accessing List Enumerations, 426  
    Data Table Column, 243  
    Node, 93, 129, 131, 258  
    Objects  
        Resources, 234  
        Standard Property, 423  
        String, 422  
        Strings, 158  
    Lookup Table, 309

## M

Material. *See* Elements Material  
Materials Handling, 210  
Math  
    Epsilon, 271, 344  
    If, 144  
    Max, 179, 440  
    Min, 178  
    Remainder, 61, 315  
    SumofSamples, 157  
Model, 24  
    Model Properties, 166, 168, 325, 347, 379  
ModelEntity, 24, 25, 65  
    Batch Members Queue, 95

Initial Desired Speed, 64  
 Initial Sequence, 69  
 TimeCreated, 110  
**TimeInSystem**, 110  
**Models**  
     Airport, 38  
     Airport Revisited, 49  
     Assembly of Circuit Boards, 89  
     Call Centers, 304  
     Cellular Manufacturing, 324  
     Circuit Board Assembly Reconsidered, 105  
     Delay Object, 347  
     DMV, 119  
     Gas Station, 385  
     Ice Cream Store, 25  
     Kitting Process, 151  
     Manufacturing Cell, 64, 210  
     More Vet Clinic Operations, 256  
     Process Planning/Project Management, 419  
     Simple Tank, 378  
     Supply Chain  
         Advanced Model, 399  
         Simple, 174  
     TransferLine, 225  
     Warehouse Pickup, 337  
**Monitor**, 169  
     Crossing State Change. *See Elements Monitor*

## N

**NHPP.** *See Rate Table*  
**Node**  
     ParkingQueue, 213  
     TransferNode. *See TransferNode*

## O

**OptQuest**, 188, 318  
     Constraints, 192, 194, 322  
     Pattern Frontier, 193  
     Responses, 320  
         Thresholds, 192  
**Output Statistic.** *See Elements:Output Statistics*

## P

**Panels**  
     Navigation Panel, 24  
     Standard Library, 25  
**Path**, 66  
     Drawn to Scale, 39  
     Drawn To Scale, 67  
     Selection Weight, 59  
**Paths**, 50  
     Bidirectional, 210  
     Selection Weight Property, 50  
**Pivot Grid.** *See Experiment-Pivot Grid*  
**Process**  
     Tokenized, 136  
     Triggering Event Condition, 149  
     Triggering Event Name, 149  
**Process Steps**  
     Assign, 107, 108, 153, 227, 238, 255, 354, 387, 407, 409  
     Change Path Direction, 421  
     Conveyor DesiredSpeed, 227  
     Multiple, 168, 181, 227, 411  
     Consume, 168  
     Create, 181, 267, 409, 426, 427  
     Decide, 110, 240, 245, 406, 409, 416  
         Condition Based, 112, 181, 227  
     Delay, 113, 114, 227, 393  
         Zero Time Event, 344  
     Destroy, 112, 181, 410, 411, 426  
     End Run, 146, 149, 437  
     End Transfer, 360, 405, 425  
     Execute, 310, 342, 381, 406, 409  
         Addon Process Trigger, 365  
     Fail, 340  
     Fire, 390  
     Insert, 138, 331  
     Interrupt, 393  
     Notify, 437  
     Produce, 168, 171  
     Release, 113, 114, 408  
         Reserving /UnreservingCapacity, 289  
         To List, 236, 242  
         Worker, 260  
     Remove, 138, 313, 331, 392  
     Repair, 340  
     Reserve, 288  
     Ride, 285  
     Route, 348  
     Search, 283, 391, 432  
         ObjectList, 252  
         Properties Explained, 251, 283  
         Queue State, 312  
         SeizedObjects, 284  
     Seize, 113, 114, 406  
         From List, 235, 241  
         Number of Objects, 241  
         Worker, 260  
     Set Node, 416  
     Set Row, 153, 238, 248  
     Tally, 110, 112, 350, 406, 422, 432  
     Transfer, 182, 267, 360, 392, 410, 426  
     UnBatch, 112  
     Unreserve, 288  
     VisitNode, 351  
     Wait, 389  
**Processes**  
     Add-On Process Trigger  
         Combiner  
         Before Processing, 114  
     Monitor  
         On Event, 380, 431  
     Node  
         Entered, 228, 355  
         Exited, 228, 322, 355  
     Path  
         Entered, 228

**ReachedEnd**, 170  
**Trailing Edge Entered**, 124  
**Server**  
 Before Processing, 235  
 Entered, 107, 310  
 Exited, 108  
 Failed, 229  
 Processed, 178, 236, 246, 260, 326, 339, 390  
 Processing, 325, 339, 389  
 Repaired, 229  
**Sink**  
*Destroying Entity*, 111, 147  
 Entered, 388  
**Source**  
 CreatedEntity, 153, 238, 387  
 Exited, 106  
**SupplyInv**  
 ReorderExited, 416  
**WorkStation**  
 FinishedGoodOperation, 182  
**Add-On Process Triggers**  
 After Processing, 113  
 Before Processing, 113  
 Exited, 113  
 Processing, 113  
**Create Process**, 108, 114  
**Delay Object**  
 OnEntered, 360  
 OnExited, 365  
**InventoryReview**, 181, 408  
**Junction Object**  
 OnEnteringProject, 425  
**On\_Off\_Entered**, 226  
**Release Associate**, 339  
**RenegeProcess**, 312  
**RespondToTimerEvent**, 167, 315  
**Seize Associate**, 339  
**SetTimeClock**, 354  
**Supplychain Object**  
 OnEnteredOrdering, 405  
 OnExitedOrderingProcess, 408  
**Modifying**  
 Override, 350, 408, 410, 411  
 OnReachedEnd, 421  
**OnEnteredInputBuffer**, 341  
**OnEnteredProcessing**, 342  
**OnRunInitialized**, 178, 381  
 Adding, 178, 381  
**TimePath**  
 All Defined, 421  
**Tokenized**, 115, 170  
**TransferNode**  
 All Defined, 348  
**Triggering Event**, 168  
**Trigging Event**, 181  
**Project Management**  
 Earliest Finish, 419  
 Latest Finish, 419  
 Percent CP(Critical Path), 419  
 Slack, 419  
**Properties**  
 Element Reference, 363  
 Object Reference, 243  
*Repeat Group*, 368  
 Standard Property, 56  
**Property**  
 Create New Property, 44  
 Definition, 58  
 Set Referenced Property, 44  
 Spreadsheet View, 226  
 User-defined, 94

**R**

**Random**  
 Discrete Distribution, 55  
**Ranking and Selection**, 186  
*Rate Table*, 306  
**Referenced Property**, 95  
**Reliability**. *See Failure*  
 Reneging, 391  
**Repeating Property Editor**, 92  
 Bill of Materials, 164  
 Release, 236  
 Seize, 235  
 State Variable Assignment, 96  
**Resource**, 144  
 Ranking Rule, 144  
 Reserving, 288  
**Results**, 35  
 Categories, 36  
 Capacity, 36  
 Content, 36  
 FlowTime, 36  
 HoldingTime, 36  
 ResourceState, 36  
 Throughput, 36  
 Pivot Grid/Table, 35, 43  
 Resource utilization, 36  
**Run**  
 TimeNow, 61, 315  
**Run Setup**, 31

**S**

**Schedules**. *See Work Schedules*  
**Secondary Resources**, 274  
**SeizedResources**  
 All Functions, 246  
 LastItem, 273  
 NumberItems, 247  
 Used in Search Step, 284  
**Separator**, 104  
**Server**, 26, 65  
 Add-On Process Trigger. *See Process Add-On*  
**Capacity**  
 Workschedules. *See WorkSchedules*  
 Capacity Type, 68  
 Dynamic Selection Rule, 145  
 InputBuffer, 267

**InputBuffer.Contents**, 31  
**Process Types**, 155, 292  
**Processing.Contents**, 31  
**Reliability Logic**, 229  
**Resource States Explained**, 120  
**States Explained**, 119  
**Static Ranking Rule**, 253  
**Station**  
  **InputBuffer**, 119  
  **OutputBuffer**, 119  
  **Processing**, 119  
**Stations Explained**, 119, 341  
**Subclassing**. *See Subclassing*  
**Taken Apart and Features Explained**, 340  
**Task Sequence**. *See Task Sequence*, *See Task Sequence*  
**Zero-Time Processing Pickup Point**, 221  
**Setup Times**  
  **Sequence Dependent**. *See WorkStation:Setup*  
**SIMIO**  
  Start page, 22  
**Simulation**  
  **EndingTime**, 52  
  **Starting Time**, 52  
**Simulations**  
  **Input Analysis**. *See Experiment:Input Analysis*  
  **Output Analysis**, 197  
  **Steady State**, 197  
  **Terminating**, 197  
  **Warmup period**, 201  
**Sink**, 26  
  **Add-On Process Trigger**. *See Processes Add-On*  
  **Statistic Collection**, 104, 310  
**SMORE**, 148, *See Experiment SMORE*  
**Source**, 26, 65, 141  
  **Arrival Mode**  
    **Arrival Table**. *See Appointment Schedules*  
    **Interarrival**, 29, 39  
    **On Event**. *See Source Arrival Mode*  
    **Time Varying Arrival Rate**, 53, 306  
  **Before Exiting State Assignments**, 55  
  **Before Exiting State Assignments**, 96  
  **Initial Capacity**, 44  
  **Maximum Time**, 146  
  **Single Source vs. Multiple Source**, 71  
  **Source Object**, 29  
  **State Assignments**, 106, 176  
  **Stopping Conditions**, 146  
  **Table Reference Assignment**, 74  
  **Time Varying Arrival Rate**, 53  
  **Standard Property**, 56  
    **Integer**, 56  
    **State**, 367  
    **String**, 237  
  **Starting Time**. *See Simulation:Starting Time*  
  **State Assignments**. *See State Variable*  
    **Creating your own**, 366  
  **State Statistics**. *See Statistics:State*  
  **State Variable**, 54, 132  
    **Before Existing State Assignments**, 176, 412  
    **Definition**, 58  
    **Discrete State**, 91  
      **Integer**, 152  
      **String**, 152  
    **On Entering State Assignments**, 102  
      **String**, 132  
      **Vector**, 132  
    **State Variables**  
      **Vector**, 81  
  **Station**  
    **EntryQueue**, 358  
**Statistics**  
  **Observation-Based**, 102  
  **Results/Reports**. *See Results*  
  **State**, 79  
  **State Statistic**, 102  
  **Tally**, 79, 102, 123, *See Elements Tally*  
  **Time-persistent**, 102  
  **Status Labels**. *See Animation*  
  **Status Tables**. *See Animation*  
  **Storage**, 138  
  **String**  
    **Format**, 182, 353  
  **Subclassing**, 340  
    **Delay Object**, 347  
    **Derived/Specialized**, 347  
    **Server**, 340  
    **TimePath**, 420  
    **TransferNode**, 347, 428  
    **Workstation**, 400  
  **Symbol**  
    **Create New**, 90

## T

**Tables**  
  **Data Table**, 56, 73, 237, 247, 314, *See*  
    **Appointment Table**, 140, 150  
    **Arrival Table Absolute**, 140  
    **Arrival Table Relative**, 150  
    **Automatic Creation of Element Columns**, 85  
      **Initial Property Values**, 85  
    **Automatic Row**, 73  
    **Random Row**, 74  
    **Related Table**  
      **Foreign Key**, 76  
      **Set Column as PrimaryKey**, 76  
  **Related Tables**  
    **Foreign Key**, 162  
  **Related Tables**, 75, 162  
  **Related Tables**, 247  
  **Resource List Column**, 243  
  **Row Specification**, 57  
  **Search**. *See Process Steps Search*, *See Process Steps*  
    **Search**  
    **Tally Statistic Element Reference**, 82, 83  
  **Data Tables**  
    **Child Table**, 161  
  **Related Table**  
    **Foreign Key**, 75  
  **Releated Table**

Primary Key Columns, 75  
 Sequence Table, 68, 72
 

- By Sequence TransferNode, 70
- Related Table, 75

 Table Reference Assignment, 74  
 Time Indexed, 316
 

- TimeIndexedRowFunction, 316
- TimeIndexedValueFunction, 316

 Tally Statistics. *See* Statistics:Tally  
 Tank, 382, 383
 

- Tank Empty Add-on Process Trigger, 385

 Task Sequence
 

- Precedence Rules, 303

 Task Sequence, 155, 292
 

- Branch Types, 296

 Process Type
 

- Process Name, 298
- Specific Time, 298
- SubModel, 298

 Task Sequences
 

- Task Precedence Method, 156
  - Immediate Predecessors Method, 156
  - Immediate Successors Method, 156
  - Sequence Number Method, 156

 TimePath, 26, 93, 352
 

- Random Time, 152, 414
- Reached End Process Trigger, 170
- Travel Time, 95

 Timer, 167  
 Timer Element. *See* Elements Timer  
 Token
 

- Add New, 136
- Adding New, 116
- Associated Entity, 282
  - Multiple Tokens, 310
- Creating New Ones, 322
- Custom, 115
- Explanation, 105
- Return Value, 252, 283
- Returned from Search, 251, 283

 tornado chart. *See* Experiment:Input Analysis:Tornado Chart  
 Transfernode
 

- Reserving the transporter for later use, 288
- Routing
  - Blocked Destination Rule, 261
  - Choosing From List, 261
  - Selection Condition, 261
  - Selection Expression, 261
  - Selection Goal, 261

 TransferNode, 42, 51, 59, 210
 

- All Processes, 348
- BySequence, 70
- Decision Point, 61
- Definition, 66, 70
- Entered Add-On Process, 228
- Entity Destination Type, 70, 93
- Entity Destination Type- Select from List, 94
- Entity Destination Type-Shortest Path, 131

 Events, 349  
 Exited Add-On Process, 228  
 External View – Output Node, 327, 404  
 Inherited Properties, 348  
 Outbound Link Rule, 59  
 ParkingStation, 348  
 Ride On Transporter, 211, 218, 278  
 Ride Process Step, 285  
 Selection Expression, 94  
 Selection Goal, 131  
**Setting Traveler Capacity**, 135  
 Subclassing, 347  
 TransferNode Properties, 51  
 Trimble 3D Warehouse, 39

## V

Vehicle, 211, 259  
**Fixed Route**, 222  
**Free Space Travel**, 223  
 Idle Action, 212  
 Initial Desired Speed, 212  
 Initial Node(Home), 212  
 ParkingStation.Contents, 214, 215  
 Ride Capacity, 212  
 RIDE STATION queue, 214  
 Task Selection Strategy, 288  
 Vehicles, 210  
 View, 31
 

- 2-D, 31
- 3-D, 31

## W

Work Schedules, 60, 67, 306, 334
 

- Day Patterns, 60
- Start Date Explanation, 88
- Using Properties, 318

 Worker, 260
 

- Dynamic Selection Rule, 266
- Transporter, 276

 Workstation, 151, 152, 175
 

- Changeover Matrix, 159
- Operation Quantity, 160
- Processing Batch Size, 160
- Setup Time
  - Fixed, 155
- Setup Times
  - Sequence Dependent, 158
  - Random, 161

 Subclassing, 399  
 Teardown, 155

## Z

Zero-Time Events. *See* Event-Zero Time

